A Genetic Algorithm for UAV Routing Integrated with a parallel Swarm Simulation

THESIS

Matthew A. Russell, Second Lieutenant, USAF

AFIT/GCS/ENG/05-16

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/05-16

A Genetic Algorithm for UAV Routing Integrated with a parallel Swarm Simulation

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Matthew A. Russell, B.S.C.S.

Second Lieutenant, USAF

March 2005

AFIT/GCS/ENG/05-16

A Genetic Algorithm for UAV Routing Integrated with a parallel Swarm Simulation

Matthew A. Russell, B.S.C.S.

Second Lieutenant, USAF

Approved:

/signed/	4 Mar 05
Dr. Gary B. Lamont (Chairman)	date
/signed/	4 Mar 05
Dr. Meir Pachter (Member)	date
/signed/	4 Mar 05
Dr. Gilbert L. Peterson (Member)	date

Dedication

IN the beginning God created the heavens and the earth.

And God said, “Let there be light,” and there was light. And God Said, “Let there be an expanse between the waters to separate water from water.” And it was so. And God said, “Let the water under the sky be gathered to one place, and let dry ground appear.” And it was so. Then God said, “Let the land produce vegetation.” And it was so. And God said, “Let there be lights in the expanse of the sky.” And it was so. And God said, “Let the water teem with living creatures, and let birds fly above the earth across the expanse of the sky.” And God saw that it was good. And God said, “Let the land produce living creatures according to their kinds.” And God saw that it was good.

Then God said, “Let us make man in our image, in our likeness, and let them rule over the fish of the sea and the birds of the air, over the livestock, over all the earth, and over all the creatures that move along the ground.” So God created man in his own image, in the image of God he created him; male and female he created them.

This is the account of the heavens and the earth when they were created.

Table of Contents

Page

Page

AFIT/GCS/ENG/05-16

Abstract

This research investigation addresses the problem of routing and simulating swarms of UAVs. Sorties are modeled as instantiations of the NP-Complete Vehicle Routing Problem, and this work uses genetic algorithms (GAs) to provide a fast and robust algorithm for a priori and dynamic routing applications. Swarms of UAVs are modeled based on extensions of Reynolds swarm research and are simulated on a Beowulf cluster as a parallel computing application using the Synchronous Environment for Emulation and Discrete Event Simulation (SPEEDES). In a test suite, standard measures such as benchmark problems, best published results, and parallel metrics are used as performance measures.

The GA consistently provides efficient and effective results for a variety of VRP benchmarks. Analysis of the solution quality over time verifies that the GA exponentially improves solution quality and is robust to changing search landscapesmaking it an ideal tool for employment in UAV routing applications. Parallel computing metrics calculated from the results of a PDES show that consistent speedup (almost linear in many cases) can be obtained using SPEEDES as the communication library for this UAV routing application. Results from the routing application and parallel simulation are synthesized to produce a more advanced model for routing UAVs.

A Genetic Algorithm for UAV Routing Integrated with a parallel Swarm Simulation

I. Introduction and Motivation for Unmanned Aerial Vehicle Research

If we knew what it was we were doing,
it would not be called research, would it?
–Albert Einstein

American service men and women have witnessed and contributed to some of the most phenomenal technological advances in history. The invention of wireless radios, modern telephony, and satellites have changed communication channels from slow inefficient postal deliveries during the days of the American Revolution to lightning fast electronic message deliveries in the present day. The invention of the airplane allows us to deploy with almost no notice to hostile areas on the other side of the globe and neutralize our enemies almost overnight; prior to the turn of the last century, it would have taken weeks or months to plan, mobilize, and neutralize threats to our American way of life. Now, in the 21st Century, Unmanned Aerial Vehicles (UAVs) are saving and will continue to save even greater numbers the lives of brave men and women that would gladly risk their own lives by engaging enemies across the 'wild blue yonder.'

By the year 2010, a third of the military fleet may be remote-controlled [50]. With the continual advances of Artificial Intelligence (AI) methods such as machine learning, the proportion of remote-controlled fleet will decrease in lieu of Unmanned Aerial Vehicles (UAVs)–'smart' planes capable of flying in a near-autonomous mode with a man-in-the-loop controller only present to handle exceptional circumstances. Already, the Predator spy plane can cruise at a speed of 84 mph, accomplish a picture perfect landing, and communicate high resolution satellite images and video feeds in realtime. In addition to realizing the potential of UAVs as a suitable platform for reconnaissance applications, the military's high level leadership is also realizing their potential for offensive weapons capability.

During a recent engagement in Afghanistan, a Predator built by General Atomics Aeronautical Systems launched a Hellfire missile and destroyed a Taliban convoy [50], marking the first time a UAV successfully struck a target with a weapon. By 2010, the unmanned aircraft market could exceed 6 billion dollars (up 1 billion from 2001). An estimated 3 billion dollars of the Defense Department's budget is dedicated to developing and buying UAVs [50]. UAVs are small, quick, less expensive than their manned alternatives, and can significantly simplify some air campaigns; UAVs allow commanders to take otherwise unacceptable risks because the worst possible outcome of tactical air strikes can be nearly vanquished: the loss of human life.

1.1 Current UAV Research

UAV research is the ultimate scientific engineering endeavor. The overall problem imposed by UAV research is to be able to program a plane with routing and other mission-related parameters, launch the plane, and confidently assume its safe return with satisfactory mission completion. In most circumstances, more than one plane is needed to complete the task, and so the problem of routing a fleet of planes in some type of formation is presented.

Successful mission completion for a fleet of UAVs in even the most limited theaters of war begs analysis and insight from the spectrum of scientific research. Research endeavors from psychology, human factors engineering, aerodynamics, electronic engineering, computer science, statistical analysis, navigation, biology and AI are pertinent among the gamut of others. Routing swarms of UAVs can be modeled as a control problem, a job-shop scheduling problem, a vehicle routing problem, or a quadratic assignment problem. Once an abstract model is instantiated from a compilation and pruning of current research, empiricism, statistics, and fault tolerance become pertinent.

UAV modeling and routing has recently become one of the predominant applications of applied swarm intelligence. The rationale is that since flocks of geese, herds of cattle, schools of fish, colonies of ants, and other biologically inspired models are capable of decentralized self-governance, researchers and scientists should be able to produce efficient and effective swarm models for commercial, industrial, and military applications that lend themselves to this abstract model. The swarm intelligence problem is often modeled as a dynamic sensor network that may involve database transactions; from a communication standpoint, the swarm is indeed a network of sensors communicating back and forth [40, 59]. Given the tremendous flexibility and scope of swarm intelligence and routing, the impact of pertinent research impacts the spectrum from the way our ordinary citizens live their day-to-day lives to the way our military fights its wars.

'Simple' UAVs capable of flying and collecting data with sensors are already being constructed very cheaply as part of university research and are being used as platforms for teaching control, applied AI, and robotics to students. Figure 1.1 shows a UAV about the size of a briefcase that Carnegie Mellon University uses for student research projects; it costs approximately $600 USD and is not difficult to construct [70]. The advent of such affordable technology allows universities and other educational institutions to make low-cost yet significant contributions to UAV research that otherwise would not be possible.

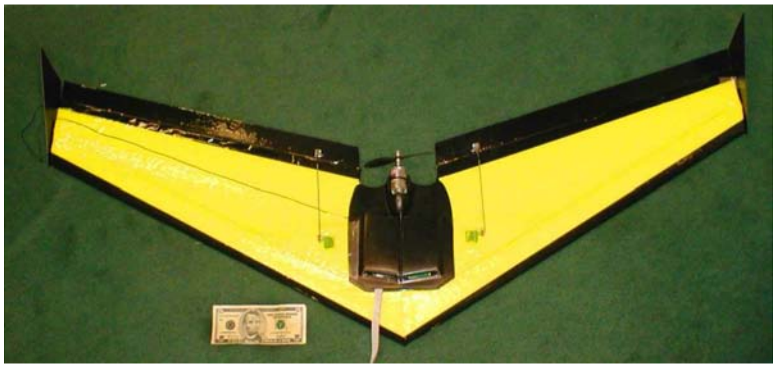


Figure 1.1: A inexpensively constructed UAV used by Carnegie Mellon University for teaching students control theory, artificial intelligence, and robotics.

1.2 Research Goal, Objectives, and Approach

Focused and well planned research is an exciting and rewarding opportunity to further one's own education and concurrently contribute to a significant needs in the world. Although the fruition of particular research efforts should be communicated in the most honest and neutral manner possible, the research process and its results can never be completely bias free. Each researcher has inherent bias and his or her own semantic network [35] of thoughts and associations that do influence the research and spur subtle spins on any particular part of the endeavor. Figure 1.2 shows just a few of the associations from a contrived semantic network belonging to an Air Force Institute of Technology (AFIT) graduate student. Although the best researchers proactively

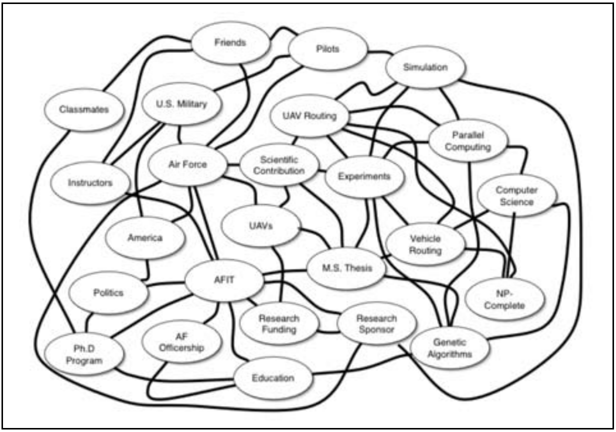


Figure 1.2: Just one sliver of a fictitious AFIT student's semantic network.

try to not let their own biases influence the outcome of research, some amount of inherent bias is inevitable, whether it be in the way a hypothesis is framed, why a particular research topic is considered important, what constraints are imposed by research funding, the nature of assumptions made about a particular problem, the rationale for using a particular problem solving approach, or why a problem is actually even considered a 'problem.' Having active awareness of research bias is an important yet very subtle and often overlooked facet that researchers and research institutions should periodically remind themselves of from time to time.

1.2.1 Goal. The overall research goal is to develop an efficient and effective UAV swarm routing algorithm and to synthesize this algorithm into existing Air Force UAV research in conjunction with AFIT and the Air Force Research Laboratory Electronic Warfare Sensors Directorate (AFRL/SNZW) Virtual Combat Laboratories (VCL). Attainment of this research goal and underlying supporting goals is measured by successful completion of the following high level objectives:

Objective 1: Formulate, implement, and empirically measure the performance of a swarm routing algorithm by means of efficiency and effectiveness metrics

Fulfillment of this objective produces an algorithm that is capable of providing a high quality 'online' solution to a UAV swarm routing problem. Applications of UAV swarm intelligence initially require high quality *a priori* solutions to complex routing problems, and also require that these solutions be robust to changes in the routing schedule that often occur from a dynamic threat environment, changes in mission planning, and other factors related to the fog of war.

Objective 2: Analyze, evaluate, and the improve the efficiency of the existing AFIT Swarm Simulator (AFITSS) as a Parallel Discrete Event Simulation (PDES) and a swarm intelligence model

Prior to this reearch, the AFITSS exhibits unacceptable parallel performance that has not yet been satisfactorily explained or improved. It is not able to simulate many more than 100 UAV swarm members in parallel without unacceptable run times and tremendous parallel computing inefficiencies. Thus, it must undergo additional parallel performance experimentation and analysis in order to adequately visualize and simulate large swarms of UAVs in realtime or within an acceptable delta of realtime. Increasing the fidelity of the simulation is crucial to making an important contribution to AFRL/SNZW research, and to swarm simulation in general.

Objective 3: Application and in depth analysis of a Genetic Algorithm for online UAV routing

A primary outlet of existing AFIT research involves the AFIT Swarm Simulator (AFITSS), but the simulation lacks the inherent ability to route from one coordinate to another. Although the simulation does provide a realistic simulation of swarm behavior based primarily on an extension of Reynolds' model [53] by Kadrovach [39], the simulation lacks the capability to route the swarm amongst a series of waypoints. Extending the AFITSS to route swarms is absolutely essential to simulating realistic Air Force missions involving reconnaissance and bombing sorties, or virtually any other scenario of interest.

1.2.2 Research Approach. This research decomposes the UAV routing problem into two primary subproblems: 1) an accurate swarm model that can be successfully routed amongst coordinates, simulated, and visualized; and 2) a planning system that schedules way points for the planned mission, handles dynamic routing constraints, scheduling changes, etc.

An effective swarm model must be able to simulate swarm behavior as observed in natural systems in addition to routing the swarm through a series of waypoints. In order to achieve parallelism, the swarm model must also be capable of subdividing the swarm and giving each subswarm some degree of independence. An effective planning system should be able to exploit the swarm's ability to divide into subswarms based on an definable objective function such as maximizing mission success, minimizing damage inflicted, and determining the waypoint sequences.

Measurable attainment of the research objectives is accomplished via an experimentation suite with specific sub-objectives for each experiment. The AFITSS PDES augments and provides some of the visualization for this research along with Audit Trail Viewer from AFRL Rome Labs, NY. Particular emphasis on fusing Kadrovach's Ph.D research [39] and Corner's M.S. research [16] with current literature on routing delivers an integrated UAV swarm routing package that can be visualized, measured for performance, and compared to other contemporary work. Metrics for analyzing the research results include comparisons to theoretical analysis, well-known benchmark problems, and the best results found in the literature.

1.3 Overview of Thesis

Research is a nonlinear process, and a research thesis is not a sequential outline of the research effort. Therefore, this document is organized in a way that communicates the effort in clear, complete, and concise terms. This chapter along with Chapter 2 provides an introduction and broad overview of university level research along with the background for swarm intelligence, routing, algorithmic techniques, and parallel simulation. Most of the background necessary for understanding modeling concepts, experimentation, and analysis is presented in this chapter, although the reader's ability to consult provided references is assumed. Chapter 3 provides a high level modeling of the routing problem and parallel simulation, and Chapter 4 narrows its focus specifically on the routing problem. Chapters 5 and 6 relay the experimentation as a suite of 10 experiments along with the results and analysis that fulfill the research objectives. Finally, Chapter 7 provides concluding remarks, commentary on the completion of research objectives, and directions for future work.

1.4 Research Sponsorship

This research directly supports the Air Force Research Laboratory Electronic Warfare Sensors Directorate (AFRL/SNZW) Virtual Combat Laboratory (VCL), Wright-Patterson Air Force Base (WPAFB), Ohio. The VCL "provides Sensors Directorate engineers and scientists with a high-fidelity virtual network-centric sensor "workbench" for algorithm development, technology insertion, proof-of-concept demonstration, and 'system of systems' experiment so that operators can see the impact of emerging technology in a synthetic combat environment." The specific point of contact at the AFRL/SNZW VCL is Mr. Mike Foster.

1.5 Summary

This chapter provides an introduction and the motivation for Unmanned Aerial Vehicle research, which is considered by many to be the ultimate scientific engineering application. This works's research objectives are presented as the development of an efficient and effective routing algorithm that can be synthesized into existing Air Force research and visualized via a Parallel Discrete Event Simulation. This research directly supports the United States Air Force, and in particular, the Air Force Research Laboratories Electronic Warfare Sensors Directorate Virtual Combat Laboratory.

II. Background and Historical Perspectives

"Give me a place to stand,
and I will move the Earth."
–Archimedes

THE advancement of UAV research specific to online routing and parallel simulation is the overarching goal of this thesis investigation. Research is an activity in which small incremental steps over an extended time period eventually become greater than the sum of their parts. Small steps gradually increase the complexity of models until they either reach fruition or need refactoring; in either case, the result is progress. This chapter establishes background and historical perspectives for UAV applications, routing large fleets of vehicles, swarms, evolutionary techniques for solving hard combinatorial optimization problems, and parallel simulation. Properly leveraging these elements is vital to the development of an efficient and effective methodology. The following chapters build upon this foundation by proposing and refining models from these perspectives in order to provide an instantiated solution model.

2.1 Overview of the Routing Problem Domain

This research involves simulating a swarm of Unmanned Aerial Vehicles (UAVs). The problem domain for this research is an extension of previous research on the Capacitated Vehicle Routing Problem (CVRP) as well as the Vehicle Routing Problem with Time Windows (VRPTW) [9,20,37,43,48,51,62–64]. Since historical approaches to the VRP have involved dissecting the problem into the TSP and the Bin-Packing Problem (BPP), a brief summary of the TSP and BPP is appropriate before a discussion of the VRP.

2.1.1 The Traveling Salesman Problem (TSP).

The classical description of the TSP involves minimizing the fitness evaluation for a permutation of cities [2]. A salesman must travel to each city only one time and return to the starting location. More formally, the objective of the

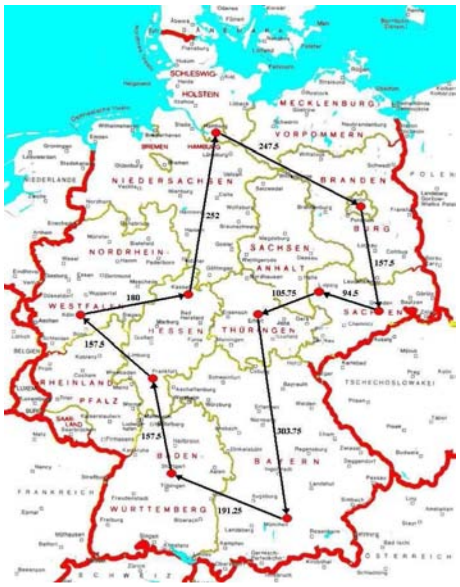
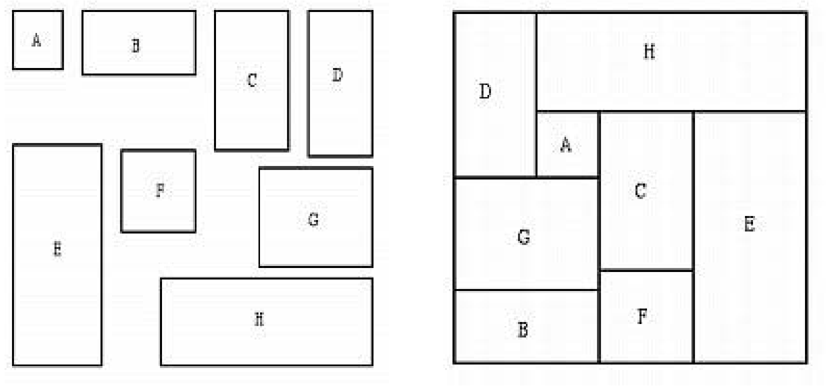


Figure 2.1: An example of a TSP for selected cities from Germany.

problem is to minimize the resulting edge costs for a Hamiltonian circuit of a fully connected graph representing the TSP. Figure 2.1.1 illustrates the optimal solution for a trivial TSP problem for cities in Germany. Some variants of the TSP include TSP with Time Windows (TSP-TW) and the Assymetric TSP (ATSP). The former increases the complexity of the basic problem by associating a minimum and maximum visitation time for each of the cities, while the latter increases the complexity by removing the symmetric property of the edges in the graph. In the real world, it is not uncommon for TSPs to involve both of the previous constraints as well as others. In all, for any graph $G = (V, E)$, there are exactly $|V|!$ permutations of the vertices. Stirling's approximation shows that an upper bound of the problem is then $O(n^n)$, where $|V| = n$ [34]. The TSP is shown to be NP-Complete [30]. Many references in the literature are available for a complete discussion of NP-Completeness [15, 34].

It is interesting to note that an application of the TSP involves using robotic arms to solder points on circuit boards, not actually traveling to cities. This is quite reasonable considering that most circuit boards have thousands and thousands of soldering points, while traveling professionals seldom travel through more than scores of cities at any given time. The TSP is also applicable



(a) BPP Problem (b) BPP Solution

Figure 2.2: BPP Example

to UAV swarm routing; a common scenario may be the case where a single swarm of UAVs must collect sensor data from a number of targets. Minimizing the travel time between locations is prerequisite to maximizing reconnaissance time.

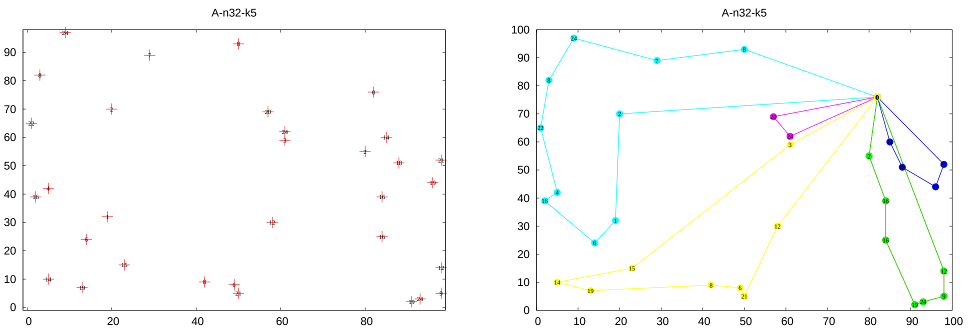
2.1.2 The Bin-Packing Problem (BPP). The classical description of the BPP involves minimizing the number of fixed space bins for which a given number of fixed space items are to be inserted. More formally, the objective is to find a partition of a set of objects subject to an assignment constraint such that the cardinality of the partition is minimized. The complexity of the problem is increased if bins and/or items of heterogeneous size are considered. Figure 2.2 illustrates a typical BPP. Much like the TSP, all possible permutations of partitions must be attempted in order to guarantee an optimal solution, so for only two bins and n objects, $(n-1)!$ permutations of partitions of objects exist. Again using Stirling's approximation, a lower bound of $O((n-1)^{n-1})$ is an upper bound on the time complexity of the problem [34]. The TSP is shown to be NP-Complete [30].

Bin-packing has applications in both two- and three-dimensional space; common applications of the bin packing problem involve truck loading, storing objects in facilities, and recycling glass [1].

2.1.3 The Vehicle Routing Problem. The Vehicle Routing Problem (VRP) is one of the most common problems in practice and dates as early as organized civilization in the form of goods distribution and trading. The VRP addresses the issue of how to route a fleet of delivery vehicles through several circuits of destinations, where each destination requires a specific demand of some product that is transported via the vehicles. The vehicles start and end routes at one or more depots, and it is desirable to minimize the total cost of accomplishing the deliveries. This total cost is a function of the total number of vehicles needed to accomplish the deliveries as well as the total distance the vehicles travel. The problem is subject to a plethora of constraints including vehicle types, delivery time windows, minimizing total cost, time travelled or the number of vehicles, not visiting any destination more than one time, and asymmetric routing. Because the VRP is a 'hard' combinatorial problem[1] whose complexity exceeds that of other hard problems such as the traveling salesman problem (TSP), brute force methods are not appropriate. In fact, the largest solvable instances of the VRP are two orders of magnitude smaller than those of TSP [52]. Figure 2.3 illustrates an 'easy' VRP benchmark problem and its proven optimal solution.

The VRP involves both TSP and BPP. In fact, a straight forward approach to the problem involves first solving the TSP and then applying BPP on the resulting Hamiltonian circuit. This strategy, known as 'route-first', is to route the vehicles first in order to minimize the total distance travelled and then cluster the delivery locations in such a way as to minimize the total number of vehicles later. Another strategy, known as 'cluster-first', is to cluster the delivery locations first using the minimum number of vehicles possible and then find the routing that minimized distance travelled. Research in 'cluster-first' strategies has been predominantly more common in the past twenty years because minimizing the number of vehicles has been deemed a higher priority constraint than minimizing the total distance travelled. This is not uncommon if one considers the cost of purchasing additional vehicles in order to meet problem constraints relative to the

[1]The convention used in this document is that a 'hard' combinatorial problem is the optimization version of a decision problem proven to already be in the class of NP-Complete problems.



(a) Benchmark Problem (b) Optimal Solution

Figure 2.3: Benchmark problem A-n32-k5 and its optimal solution.

cost of purchasing additional fuel and additional manning of vehicles in order to meet problem constraints. Additionally, 'cluster-first' strategies have produced better results than 'route-first' strategies, although 'route-first' strategies perform well in an asymptotic sense [2].

Future military objectives that involve routing swarms of micro-UAVs, might deem any number of constraints to be of higher priority than minimizing the swarm size, especially as cost of manufacturing micro-UAVs decreases and they become an expendable resource. Each particular routing algorithm like so many other algorithms depends upon the problem domain. It is important to note, however, that solving both the TSP and the BPP problem optimally back-to-back does not guarantee an optimal solution to the VRP. In the classical description of the Capacitated VRP, a fleet of homogeneous vehicles exists with each member of the fleet having a fixed capacity. A number of locations also exist and each location has some demand for a given item. The objective of the problem is to minimize the overall cost of supplying the locations via the delivery fleet. The objective function of the problem widely varies and is specific to the particular application.

As applied to UAV routing, the CVRP is formulated by the following precise mathematical description, where L is the set of all targets of interest, V is the set of all UAVs, Q is the set of traveling costs, G is the set of routes, δ is a distance function, γ is a capacity function, and α is a demand function:

Definition 2.1. *UAV Routing Problem:*
Given:
$L : \forall l_i \alpha(l_i) \geq 0, i > 0; \alpha(l_0) = 0$
$V : \forall v_i \ \gamma(v_i) = k, \ k > 0$

Compute:
$Q : q_{ij} = \delta(l_i, l_j)$
$G : g_k = \{l_0 \bigcup L \times_k L \bigcup l_0\}$

Constraints:
$\sum_{l \in G} \alpha(l) = \sum_{l \in L} \alpha(l)$
$\bigcup_{l \in G} = L$
$\bigcap_{g \in G} = l_0$

Objective:
$min \sum_{k=1}^{|Q|} Q_k$

Given that the VRP is at least as complex as either the TSP or BPP, a lower bound for the problem is $\Omega(n^n)$. The TSP and BPP are already intractable problems; adding in the additional constraints of time windows for each delivery location, heterogeneous vehicle fleets, or a dynamic delivery environment clearly does not simplify the problem.

2.1.4 Summary of Various VRP Instantiations. Specific instantiations of VRP include the Multiple Depot Vehicle Routing Problem (MDVRP), the Vehicle Routing Problem with Time Windows (VRPTW), the Dynamic Vehicle Routing Problem (DVRP), the Capacitated Vehicle Routing Problem (CVRP), and the Fleet Size and Mixed Vehicle Routing Problem (FSMVRP). Other variations exist, but these are the most common found in the literature [68, 69].

In fact, it is possible to construct a VRP hierarchy such that the root of the hierarchy is an instantiation of all of the previous problems. Thus, the previously mentioned problems become a simplification of the root problem. For example, the CVRP may be expressed as a specific instantiation of all of the previous VRPs with time windows $t_{min} = 0$ and $t_{max} = k$, a single depot, a homogeneous vehicle fleet, and all deliveries known *a priori*. For this reason, precisely defining the problem definition in concise mathematical notation is crucial.

2.1.5 VRP Application To UAV Operations.

Consider an environment, however, in which a given number of locations require a payload of reconnaissance or weapons munition. A very large heterogeneous swarm of micro-UAVs capable of delivering both the munitions and the reconnaissance payload exists. Given a large set of these waypoints, threats to avoid, time windows, a changing threat environment, and the potential loss of members of the swarm along the way, routing the swarm through the environment is a very complex instance of the VRP. Clearly, finding good approximation techniques to complex instances of VRP has practical interest for both industry and military applications. To solve such problems satisfactorily, the appropriate selection of an algorithm is essential.

2.2 Overview of the Algorithm Domain for the Routing Problem

Having much applicability to everyday commerce and transportation, the VRP has been approached from nearly every angle possible. Exact approaches, iterative heuristics, meta-heuristics, and stochastic search have all been applied and with varying degrees of success. Some of the most intense studies have included branch-and-bound techniques, tabu search, and iterative savings-based heuristics. Metaheuristics such as ant algorithms, genetic algorithms (GAs), and other evolutionary approaches, however, are becoming common and with much success.

Some specific approaches from the literature include Ant Colony Optimization [29, 42], Tabu Search [61], Evolutionary Strategies [37], parallel approaches [4, 48], Simulated Annealing [20], problem simplification [75], fuzzy logic [47], and branch-and-bound techniques [52]. Genetic Vehicle Representation (GVR) [43, 62, 63] is a fairly recent approach with special emphasis on the data representation. The next section discusses the following facets of evolutionary computation and how their relevancy to the VRP: genetic algorithms, artificial immune systems, artificial neural networks, fuzzy logic, ant colony optimization, and Reynolds' 'boids' research.

2.2.1 Evolutionary Computation. As early as the 1950s, computational scientists began to apply neo-Darwinian theories to the process of solving problems via Evolutionary Algorithms (EAs) [5]. Assuming the process of evolution occurred[2] and resulted in the creation of complex and powerful organisms through a process of gradual improvement over many years, they conjectured that a similar process for approximating solutions to complex problems may work, and so the field of Evolutionary Computation (EC) was defined[3].

Generally speaking, EC and EA approaches capitalize on the good building blocks in partial solutions that propagate exponentially through many generations until a near optimal total solution is attained. The conceptual steps to an EA are quite simple. The first step of an EA is to initialize a population of individuals such that each individual is encoded with a random solutions to a problem. Individuals from the population recombine and produce offspring with some probability, can be mutated with some probability, and then applying Darwin's clich*é* of "Survival of the Fittest", only some number of the fittest individuals survive the process of selection and survive to reproduce in the next population. This process continues for a given number of generations or until a certain threshold value is met. The fittest individual existing in the final population then provides the approximated solution to the problem. The most common forms of EC include Genetic Algorithms (GAs), Evolution Strategies (ESs), Genetic Programming (GPs), and Evolutionary Programming (EP) [5]. Early research in the field started with the manipulation of finite state machines with EP, and the process grew into wider acceptance via ESs in Europe and GAs in the United States.

[2]Objection to evolution as a scientific truth is reasonable, and the success of EAs as applied to complex optimization problems is not supporting evidence for evolution. Interestingly, one need not affirm the *t*heory of evolution in order to benefit from the robust and powerful problem solving characteristics of EAs. This note is worth clarifying, since virtually all literature on the subject presupposes evolution as a scientific truth, even though doing so is neither a necessary nor sufficient condition for demonstrating the success of EAs.

[3]The actual term 'Evolutionary Computation' was not coined until around 1991 [5]

2.2.2 Genetic Algorithms. GAs were first proposed by John Holland in 1975 [36,72]. The three features that first distinguished GAs from other EC techniques were the bitstring representation, the proportional selection operator, and the crossover operator. The crossover operator, in particular, is what makes GAs distinctive from other techniques [5]. The basic algorithm for a GA consists of repeating the following steps: (i) initializing a population of solutions at time t, (ii) evaluating the population with an objective function and scaling to produce fitness values, (iii) selecting individuals for recombination based upon fitness values, and (iv) recombining individuals with one another to produce a new generation of individuals at time $t + 1$. In general, the algorithm repeats steps (ii)-(iv) for some number of generations, although other stopping criteria such as thresholding functions can also be used.

2.2.2.1 Schema and Good Building Blocks. GAs rely on the 'Schema Theorem', which asserts that when using the proportional selection operator, the number of good building blocks increases exponentially as the number of generations increases. [72] This approach relies upon the assumption that 'good' building blocks start small and incrementally build larger good solutions[4]. Some GA approaches produce a new population of individuals at each generation solely by recombination, while others produce new populations by recombination as well as allowing parents from generation at time t to survive into the subsequent generation. The former strategy is known as (μ, λ), while the latter is known as ($\mu + \lambda$). Generally speaking, the former produces more exploration, while the latter produces more exploitation. One must be cautious when using a (μ, λ) strategy, because good building blocks can be harder to keep from population to population [5].

GAs are conceptually simple, although embedding specific problem domain constraints and guaranteeing feasible solutions after recombination is usually not trivial. A formal GA adapted from Bäck that is capable of repairing infeasible solutions is given in Algorithm 1. Repairs to infeasible solutions need not be explicitly inserted back into the population. Rather, infeasible solutions can

[4]Acceptance of Schema Analysis as presented by Goldberg [31] exist in the literature and acceptance of the building block hypothesis is by no means unanimous [67].

be deterministically repaired and evaluated, while remaining genotypically infeasible. Explicitly repairing individuals is analogous to Lamarckian evolution, while implicitly repairing individuals is analogous to Baldwinian evolution [6].

In Lamarckian models of evolution, performance gains from individual learning are mapped back to the genotype, and thus, can be passed on to its offspring. [6]. In EC, the mapping back to the genotype that takes place is an explicit repair to an infeasible genotype. Baldwinian models, however, do not explicitly repair infeasible solutions. Rather, these models cling to the so-called Baldwin Effect and simply interpret the infeasible solution as a feasible one. Some experiments have shown that using Lamarckian techniques often result in convergence to local optima, while Baldwinian techniques are more likely to result in convergence to a global optimum [6].

```
Data     : $t_{max}$, constraints: $Q = \{q_1, \ldots q_k\}, p_{replace}$
Result   : $P(t_{max}) := \{\vec{a}_1(t_{max}), \ldots, \vec{a}_\mu(t_{max})\}$
function repair: $\zeta(\vec{a}''_k(t))$ :
begin
    if $\sum_{i=1}^{m} \pi(q_i, \vec{a}''_k(t)) = 0$ then
        return $\vec{a}''_k(t)$;
    end

    if $(RAND_{[0,1]} <= p_{replace})$ then
        $\vec{a}''_k(t) := \eta(\vec{a}''_k(t))$;
        return $\vec{a}''_k(t)$;
    else
        return $\eta(\vec{a}''_k(t))$;
    end
end

begin
    t:=0;
    initialize: $P(0) := \{\vec{a}_1(0), \ldots, \vec{a}_\mu(0)\} \in I^\mu$, where $I = \{0,1\}^l$;
    while $t < t_{max}$ do
        recombine: $\vec{a}'_k(t) := r'_{\{p_c, z\}}(P(t)) \forall k \in \{1, \ldots, \mu\}$;
        mutate: $\vec{a}''_k(t) := m'_{\{p_m\}}(\vec{a}'_k(t)) \forall k \in \{1, \ldots, \mu\}$;
        evaluate: $P''(t) := \{\vec{a}''_1(t), \ldots, \vec{a}''_\mu(t)\} : \{\Phi(\vec{a}''_1(t)), \ldots, \Phi(\vec{a}''_\mu(t))\}$, where $\Phi(\vec{a}''_k(t)) = \delta(f(\zeta(\Upsilon(\vec{a}''_k(t)))), P(t-\omega))$;
        select: $P(t+1) := s(P''(t))$, where $p_s(\vec{a}''_k(t)) = \Phi(\vec{a}''_k(t)) / \sum_{j=1}^{\mu} \Phi(\vec{a}''_j(t))$;
        t := t+1;
    end
    return $P(t)$
end
```

Algorithm 1: GA Algorithm with repair function

An alternative to repairing infeasible solutions due to the complexity of handling all of the violated constraints is to assign a penalty to them. Assigning a penalty to infeasible solutions is a design choice that relies upon the assumption that infeasible solutions have lower fitness values that infeasible solutions, and the search progresses toward feasible solutions. Optimal solutions, in general, lie on the boundary between feasible and infeasible solutions. Therefore, one must

Table 2.1: A brief summary of the selection mechanisms discussed

Type	Time Complexity	Pressure	Advantages	Disadvantages
Roulette Wheel	$\Theta(n)$	Relatively high	Schema analysis applies	'Super-fit' individuals can dominate
SUS	$\Theta(n)$	Relatively high	Schema analysis; efficient; hedges against 'super-fit' individuals	'Super-fit' individuals still can dominate
Tournament	$\Theta(q)$	$\propto \frac{q}{n}$	Scaling Invariant; Implementation; Convenient for parallel implementation	Loss of diversity; population representation
Rank-based	$O(n \cdot lg(n))$	Low	Scaling Invariant; Simplicity	Fitting population to distribution

cautiously choose penalty functions. Penalties too lenient result in exploration in the very large space of infeasibility, while penalties too severe result in the search stalling in local optima. Various penalty functions include static, dynamic, and adaptive penalties. [6].

2.2.2.2 Selection Operators. Common selection operators include proportional selection, tournament selection, and rank-based selection [5]. The selection mechanism employed is a significant design decision when implementing a GA. Three of the most common selection mechanisms are discussed and summarized in Table 2.1.

Proportional, commonly known as 'roulette-wheel' selection, involves choosing individuals in a population for recombination based upon their proportional fitness to one another. Usually, this is accomplished by means of calculating the overall fitness of the entire population and then comparing a particular individual's fitness to the population's overall fitness. Conceptually, each individual has a slice of a roulette wheel. The roulette wheel is spun, and the individual the wheel lands on is chosen for recombination. Individuals with very high proportional fitness commonly dominate the selection process when n separate spins are accomplished. A method entitled Stochastic Universal Selection (SUS) is a means of providing for equal representation even in the presence of 'super-fit' individuals [5]. In SUS, if k individuals are required for recombination, a single spin of the roulette wheel is accomplished and individuals are chosen based upon k equidistant pegs surrounding the roulette wheel. This method results in less-fit individuals still having a uniform probability of being chosen, whereas k consecutive spins may not. Its time complexity is $O(n)$ and one source cites it as being 'optimally efficient' [5]. Still, there is relatively high selection pressure even with SUS in proportional selection. An appropriate scaling function is another crucial factor for proportional selection since individuals chosen for survival into the next population can have very similar fitness values. Population initialization is normally assumed to be random with proportional selection.

Tournament selection involves randomly selecting individuals q at a time. The individual with the highest fitness among the q of them is chosen for recombination. Tournament selection has the advantage of being implemented very efficiently and the time complexity of only $O(n)$ [5]. Tournament selection is scaling and translation invariant. Therefore, scaling methods as used in proportional selection are not needed. Population initialization is normally assumed to be Gaussian-distributed initial population [5]. Tournament selection's selection pressure is proportional to the tournament size q. Given a population of n individuals, the selection pressure increases as $\frac{q}{n}$ increases.

Rank-based selection involves assigning selection probabilities based upon ranking individuals according to a probability distribution. An advantage of rank-based selection includes its very low selection pressure. Unlike proportional selection, 'super-fit' individuals cannot lead to premature convergence. The most fit individual receives the same rank each time. The least fit individual receives the same rank each time as well. This property also has the advantage of being advantageous when an objective function is hard to define. A difficulty with rank-based selection is defining which probability distribution to use. Oftentimes, the fitness landscape and desired solution characteristics are unknown, so assigning a distribution is as difficult as obtaining a good solution to the problem.

2.2.3 Artificial Immune Systems (AISs) and Artificial Neural Networks (ANNs). While AISs have primarily been used in the application areas of intrusion detection and optimization, they also are appropriate for use as a pattern recognition application. Pattern recognition via ANNs and AISs is vital component of real-time UAV reconnaissance applications and has also been successfully applied to optimization problems [22, 45, 46, 55]. These concepts are a logical extension and next step to Corner's experiments in applying a parallel swarm simulation to target recognition [16].

A particular application of pattern classifiers with respect to UAVs is with regard to identifying hostile targets and identifying suspicious objects. For example, a swarm of UAVs with built in pattern classifiers might train on several patterns of known enemy data such as the profile of a partially hidden tank in the desert. Later, while the swarm is en route on a reconnaissance mission, can use these patterns for identifying suspicious areas. ANNs and AISs provide some flexibility, so the pattern encountered by the swarm need not match the training pattern exactly. These methods have the advantage of being potentially as successful as the training data; however, the lack of sufficient training data is a disadvantage.

2.2.3.1 ANNs. One work by Dasgupta compares and contrasts Artificial Neural Networks (ANNs) and Aritificial Immune Systems (AISs) [22]. Since ANNs have been used extensively for pattern recognition, they provide a good benchmark application with which to make a comparison. Both ANNs and AISs are biologically inspired mechanisms.

ANNs are inspired from the neurons of the nervous system. Conceptually, ANNs are multiple layers of perceptrons. Each perceptron receives a series of inputs and if the sum of these inputs exceed a threshold defined by a thresholding function, the output of the perceptron fires, which is in turn fed into other perceptrons. After some number of layers, the network produces an output pattern. A gradient descent technique is commonly used to designate how 'incorrect' the network's output was for the given pattern and a backpropagation technique gradually adjusts the connections between the perceptrons. Given enough training examples, the network is expected to produce the expected output for all of the training examples. Once the network is trained, the network

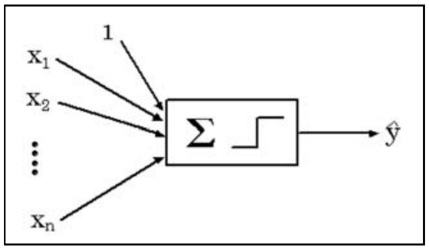


Figure 2.4: A perceptron

can be tested with additional training examples that have been previously unseen and expected to produce the desired output. Various types of ANNs exist; a few of the design decisions when building an ANN include: learning online or offline, whether or not to be recurrent, the thresholding function to use, the number of hidden layers to use, and the particular backpropagation technique to employ [46, 55]. Note that ANNs are deterministic mechanisms that produce a global behavior, generalized from sets of independent training data samples.

2.2.3.2 AISs. AISs are reminiscent on the concept of the body producing antibodies in response to antigens. The body has three primary layers of defense: the skin and mucus membranes, innate immunity, and adaptive immunity. AISs were produced from the adaptive immunological level. The adaptive immunological level's model primarily involves four types of cells: helper T cells T_h, killer T cells T_k, suppressor T cells T_s, and B cells B [21].

To illustrate the role each type of cell plays, consider the following scenario: an antigen invades the body and infects a cell. The cell recognizes the antigen as being foreign to the body, and although infected, it is capable of presenting a part of the antigen on its surface as a signal to other cells via immunoglobin receptors. T_h cells near the infected cell also recognize parts of the antigen on the cells surface and produce lymphokines, signals that tell B cells to proliferate and bind to particular sites of the antigen surface called epitopes. The activated B cells can then do two things: 1) secrete antibodies that can bind to the recognized antigens neutralize them or identify them to phagocytes, macrophages, or T_k cells and 2) undergo the processes necessary to produce memory cells that result in the mutations necessary to recognize the antigens in subsequent encounters. Macrophages and T_k cells kill the infected cells, which cannot be repaired and would otherwise continue the spread of antigens. T_s are involved in the process by acting as robust self-detectors– cells that recognize self cells that T_h cells may otherwise identify as antigens. Conceptually the T_s cells prevent false alarms.

2.2.3.3 Application of Fuzzy Logic to AISs. The robust and distributed nature of the immune system model makes it especially suitable for anomaly detection, and it has been applied extensively to the field. An application of AISs that has received significantly less attention, however, is the application of binary pattern recognition. The AIS model basically involves identifying and distinguishing between 'self' and 'non-self', so using AISs for identifying reconnaissance imagery as friend or foe seems appropriate [22]. Additionally, fuzzy logic can be applied which allows members of sets to belong partially to more than one set with varying degrees of membership [45].

The application of fuzzy logic using an immunogenic approach seems appropriate for target recognition in theaters of war. Swarms of UAVs might have certain 'innate' abilities to recognize general 'antigens' such as other unidentified flying objects in the air and large convoys of adversarial forces. The adaptive abilities to train on and recognize specific 'antigens' in a theater such as missile silos, identified enemy headquarters, and weapons production facilities. Note that the suppressor T_s cells might also play an interesting role by being trained to recognize hospitals, religious centers, and other neutral areas. Clearly, the areas of interest in such scenarios is very small compared to the entire theater area, and fuzzy logic can indicate the degree with which a specific venue belongs to either a set of self or a set of non-self.

2.2.3.4 Application of Anomaly Detection using Pattern Classifiers for UAVs. Identifying antigens for use in an AIS might first involve generating unique signatures for each of the known threats into training data sets, training the swarm on the data sets, and then testing the trained swarm on previously unseen data sets. Given a small enough threshold on the margin of error, the swarm can then be deployed into areas of interest with expectations of reasonable behavior.

Various immunogenic libraries for different theaters might be developed and loaded into swarms of UAVs on demand or as appropriate. For example, one particular data set might be of a desert landscape with various types of current generation weaponry, existing machinery, and convoys of vehicles, while another data set may be of a jungle or arctic landscape with similar features. Through a satellite uplink, the same swarm of UAVs might monitor very diverse parts of the world or countries with diverse landscapes without landing and manually swapping the data sets. This approach might also have the benefit of reducing the size of the swarm since not all data sets have to reside on each UAV.

Still another possibility involves near real-time uplinks between swarms of UAVs. In this scenario, multiple swarms of UAVs are airborne and concurrently conducting reconnaissance missions, while one highly specialized swarm might sweep through an area with high resolution imagery peripherals and generate training data sets. These data sets are downloaded to a massively parallel machine on the land for the generation of pattern recognizers, and then the pattern recognizers are uploaded to the reconnaissance swarms or a killer swarm that might use these data sets to sweep through the area and provide air strikes or some form of close air support.

Dasgupta indicates ANNs and AISs probably produce pattern classifiers equally well for many types of problems [22]. Therefore, other characteristics might be considered to determine if one is more appropriate than another for this particular application. ANNs are deterministic, while AISs are probabilistic. Both methods can be trained a priori to evaluation with test data sets, but both models also can learn online. ANNs classify patterns by training the network on the data sets, while AISs recognize patterns in the complement space as novel (self and non-self space must have been previously calculated).

Additionally, AISs can classify patterns to belong with varying degrees of membership to more than one set. ANNs lack this ability. Perhaps the most important advantage of the AIS model over the ANN model is that a decision in an ANN is a global action that requires computation on behalf of the entire ANN, while a decision in an AIS is decentralized local decision. This difference might enable swarms of UAVs using an AIS model to generate quicker responses to changing threat environments than with an ANN model. Additionally, the design of most ANNs can be a tedious

Table 2.2: Aspects of ANNs and AISs

	AIS	ANN
Probabilistic	Yes	No
Training	Online and offline training	
Classification Method	Additional training	Self/non-self space or fuzzy logic
Decisions	Global	Local

process that requires much tailoring. Some ANNs cannot classify patterns with an infinite number of training data because of an inadequate number of hidden layers, thresholding function, or initial random weightings.

AISs and ANNs provide two possible approaches to pattern and target recognition for UAV reconnaissance applications. Ant Colony Optimization is yet another model inspired from a natural system–the foraging behavior of ants as observed in their natural habitat. It offers a biologically inspired model for routing UAVs.

2.2.4 Ant Colony Optimization. Ant Colony Optimization (ACO) models became popular after Dorigo published his Ph.D dissertation on the topic in 1992 [24]. With his introduction of 'Ant System', many other researchers became interested in the idea of ants and collective intelligence systems such as insect colonies. ACO is an interesting model because it has been applied to TSP and various routing problems in previous research although the totality of its research is quite extensive. A few applications of ACO include water distribution [74], logistic systems [29], graph coloring [18], BPP and Stock Cutting Problems [42], TSP [28], telecommunication network load balancing strategies [12, 57], routing problems [58], and optimization problems [7] among others. Being highly parallelizable, various strategies for parallelization have also been introduced [11].

The ACO model mimics the foraging behavior of ants. As an ant travels from its nest to a foraging location, it leaves a trail of pheromone that other ants can detect. These ants follow the existing pheromone trail based on several factors, some of which include the relative angle between the nest and the foraging location and the intensity of the pheromone trail. The more ants that follow a pheromone trail, the stronger the trail's intensity becomes. If very few ants follow a trail, the trail is likely to evaporate to such a degree that it can no longer be detected and is less likely to be followed. Bonabeau [25] describes a binary bridge experiment in which ants were separated from a foraging location by two paths, with one path being longer than the other. Ants that left the nest and returned first must have taken the minimum path both ways and resultantly the most pheromone was concentrated on the shortest path in the shortest amount of time. As a result, subsequent ants are more likely to follow this trail both ways because its intensity is stronger than any other possibility. After a very short time period, nearly all ants travel along the shortest path.

A simplified model of foraging behavior in general may be stated as follows:

Definition 2.2. *There exists a set of ants A, a set of locations L to include a single nest l_n and a single foraging location l_f, and a set of paths $P \ni p_{ij}$ derived from $L \times L$. There exists a bijection from the set $Q \ni q_{ij}$ on P that represents the cost (usually distance) between any two locations. For a given number of iterations, ants start at l_n and move in lock-step toward completion of a cycle that includes l_f. For example, while in location l_i, an ant chooses location l_j according to a probabilistic transition function [25] that compares the intensity of the pheromone trail τ_{ij} relative to the cost of taking the path, q_{ij}.* [5] *In order to prevent the ant system from prematurely converging on a suboptimal solution, the pheromone evaporates by a factor of ρ after each iteration. The algorithm terminates after a given number of iterations.*

2.2.4.1 Using ACO as a Swarm Routing Optmization Technique. The formulation of ACO can map cohesively to many problems formulated with graphs as the primary data structure, including the VRP [11,29,47,58]. ACO as an algorithmic basis for solving VRP instances might be especially useful for the Dynamic VRP, because ACO models have been shown to quickly converge to other good solutions when disruptions occur to the current best solution [25]. A shortcoming of ACO as an optimization technique for swarms of UAVs is that it is a coarse grained search and

[5] *In some applications such as the Traveling Salesman Problem (TSP), it is necessary to complete a Hamiltonian cycle or meet other related constraints related to visitation of locations. Typically, an auxiliary memory $M \ni M \subseteq L$ is introduced for each ant. At the beginning of an iteration, $M = L$. After each move, M is updated by $M = M - l_j$, where l_j is the current location of the ant.*

has not been shown to perform competitively with other techniques for TSP instances of 30 cities or greater.

The most straightforward approach for applying ACO to the VRP involves assigning a proper subset of $Y = L_i \times D_i$ to ant a_i from A [11] and initially assigning an ant a capacity of items given by c_{ANT} An ant's objective after being 'loaded' to its capacity and assigned a set from $Y \ni y_k$ is to minimize the total cost of completing a circuit given by the l_i terms of the assignment.[6] Once an ant has visited a location l_i, the tuple $\{l_i, d_i\}$ is removed from its 'memory' m and location l_i is added to its solution g. Ants transition amongst delivery locations until their assignment $y_k = \emptyset$. If at any time before a circuit is completed, $\forall d_i \in y > c_{ANT}$, the ant immediately returns from location l_i to the depot at a cost of $q_{i,depot}$, refills its capacity to c_{ANT}, and then continues deliveries, which may or may not immediately start back at l_i.[7] Once all ants have completed their circuit, only the ant that has generated the shortest total circuit, g_{min} contributes a quantity of pheromone to all paths on the minimal path travelled[8] A generalization of this update is given by equation 2.1. The algorithm terminates after a given number of iterations and the set of circuits returned by the final iteration is the solution returned by the algorithm.

$$\tau_{ij}(t) = (1 - \rho) \cdot \tau_{ij}(t) + \tau_\epsilon \tag{2.1}$$

2.2.4.2 Explicit Mapping to Search Constructs. A coarse mapping to abstract search constructs is useful for mapping ACO techniques to the swarm routing problem.

[6]This initial reduction is helpful because ACO cohesively maps to 'ordering' problems such as the TSP, whose problem domain is essentially the same as the one ants face in nature.

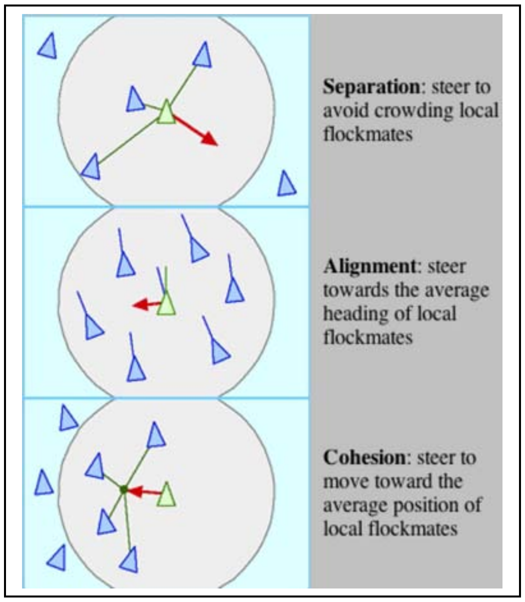
[7]Because the VRP as presented involves homogeneous delivery items, loading the delivery items poses no additional complication. Should the delivery items be heterogeneous, however, it would be necessary to process the loading of items at each return to the depot via the Bin Packing problem, another NP-Hard problem

[8]A plethora of variation on this pheromone update have been presented by various authors. The most common variations include allowing all ants to update the pheromone trail or allowing only the k best ants to contribute to updating the trail. In either case, the update may or may not be qualitatively based on their circuit traversed and/or weighted proportional to that quality.

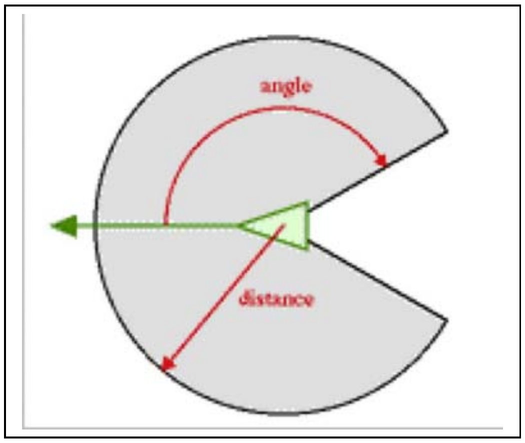
- Set of Next State Candidates–For each iteration of the algorithm, the set of next state candidates includes all possible permutations of customers assigned to ants: $Candidates = \sum_{i=0}^{ant_{max}} \{ant_i \times (L \times_m L)\}$
- Next State Generator–The next state generator randomly assigns a proportional number of customers to each ant: $Candidates = \sum_{i=0}^{ant_{max}} \{ant_i \times (L \times_m L)\} \ni m \propto \frac{|L|}{|A|}$
- Feasibility Function–The feasibility function refines $Candidates$ to ensure that no two customers are assigned to the same ant:
  $Candidates = \sum_{i=0}^{ant_{max}} \{ant_i \times (L \times_m L)\} \ni \forall i \forall j [\neg \exists \{ant_j, l_i\}, \{ant_k, l_i\}]$
- Selection Function–The selection function selects a valid mapping from $Candidates$, where $Candidates$ is generated stochastically.
- Solution Function–The solution function determines when all deliveries have been met for all customers: $\forall ant \in Candidates \forall y \in Y, y_{ant} = \emptyset$
- Objective Function–The objective function evaporates pheromone according to equation 2.1. The objective function returns the final iteration's solution.

ACO techniques almost always enhanced via a local search technique specific to the problem domain. For example, ACO-TSP provides much better results using a local search procedure such as 3-opt [25] than without it, and ACO-Bin Packing provides must better results with combined with a form of hill climbing than without it [42, 68].

2.2.5 Reynolds' Model. In 1987, Reynolds produced a simple model geared toward computer graphics that illustrated flocking behavior to a very realistic extent [53]. This model is conceptually simple, and no discussion of swarm behavior or flocking is complete without it. It places three primary constraints on the movement of each swarm member: separation, cohesion, and alignment. Figure 5(b) shows an illustration of this model as presented on Reynolds' website at `http://www.red3d.com/cwr/boids/`


Separation: steer to avoid crowding local flockmates
Alignment: steer towards the average heading of local flockmates
Cohesion: steer to move toward the average position of local flockmates

(a) Separation, Cohesion, and Alignment



(b) A swarm member's line of sight

Figure 2.5: A neighborhood and the components of the model.

Kadrovach's Ph.D research involves a centralized swarm model that is an extended implementation of Reynold's model. Corner's research parallelized this model. Chapter III discusses research by Kadrovach [39] and Corner [16,17] in detail, because it is part of the infrastructure for the current AFIT Swarm Simulator.

2.2.6 Deterministic Techniques. Deterministic techniques have received the vast majority of attention for solving the VRP. Promising branch-and-bound techniques which formulate the VRP as a Lagrangian constraint problem demonstrate impressive results [52] and much success has been obtained by Taillard with tabu search [4,61]. Deterministic approaches often rely heavily upon excellent construction heuristics, however, and these heuristics can be difficult to formulate because of problem specifics. In general, the construction heuristics attempt to group locations into sets and then provide a mechanism for ordering the set. The additional concerns of dimensionality must also be considered when choosing a deterministic algorithm. Historically, most benchmark problems having reported results with deterministic techniques are of relatively modest size.

Perhaps the first popular constructive method that relies upon a gradual construction is the well-known Clarke-Wright Savings Heuristic which dates back nearly half a century. In general, this heuristic works by first constructing n independent tours for a problem of size n. Tours can then combined iteratively according to an explicit 'combine rule' that preserves solution feasibility. The tours with the greatest 'savings' are combined. The algorithm as originally presented is found in the original paper [14]. This heuristic can be proven to construct a feasible solution to the CVRP in $O(n^2 lg(n))$ time [41], but does not make any guarantees on solution quality.

2.3 Parallel Discrete Event Simulation

Primary reasons for Parallel Discrete Event Simulation (PDES) include an enormous cost reduction in research and development and reduced execution time for models that might otherwise take very large finite time periods to execute. Fujimoto cites at least three other reasons why PDES is important. These reasons include allowing geographically distributed computers to participate in simulation, integration of simulators that execute on heterogeneous architectures, and increased fault tolerance [27].

2.3.1 Time in PDES. Perhaps the most fundamental concept in a PDES is the notion of time. Time can be regarded in at least three distinct ways: 1) Physical Time, 2) Simulation Time, and 3) Wallclock Time. Physical Time refers to the actual time in the physical system being simulated, which may stretch anywhere from nanoseconds to thousands of years. Simulation time refers to the totally ordered set of values where each value represents an instant of time in the physical system being modeled. Simulation Time (T_s) and Wallclock Time (T_w) can be derived from one another via Equation 2.2.

$$T_s = W2S(T_w) = T_{start} + Scale(T_w - T_{wStart}) \tag{2.2}$$

Wallclock time is the total time that the simulation has executed from its beginning till the present moment [27]. A clear understanding of each notion of time and what each time represents is fundamental to PDES.

2.3.2 Scheduling. At the heart of scheduling events in PDES is the *local causality constraint*. A PDES obeys the local causality constraint if and only if Definition 2.3 is satisfied.

Definition 2.3. *Local Causality Constraint - each logical process may only process events in non-decreasing time stamp order.*

Intuitively, the local causality constraint seems reasonable. In order to maintain a notion of time, events must be processed logically from start to finish. Various techniques allow for satisfaction of the local causality constraint. Four of the most common are of particular relevance.

2.3.3 Conservative Techniques. Conservative techniques address the synchronization problem by processing events only when doing so is considered 'safe.' In other words, events containing a time stamp marked t are only processed by a given process p when p can determine that it will not receive any other event with time stamp of less than t. Conservative techniques are designed such that no *rollback*s can occur. The most common conservative techniques include First-In-First-Out Queues, the Chandy/Misra/Bryand Algorithm and the Demand Driven Approach [27]. The prevalent shortcoming of conservative techniques is their inability to fully exploit the concurrency that is available in the simulation application [27]. Any possible event that might cause a violation of the local causality constraint must be prevented; thus, the simulation does not execute for some applications as fast as with optimistic techniques, which allow local causality violations but provide recovery mechanisms.

2.3.4 Optimistic Techniques. When applicable, optimistic techniques are often desirable over conservative techniques. For example, consider the microprocessor pipeline design that processes events ‘ahead of time’. Such designs can result in tremendous performance improvements. Three of the most common optimistic techniques include [27, 44]:

- Jefferson’s Time Warp
- Breathing Time Buckets
- Breathing Time Warp

Jefferson’s Time Warp processes events optimistically. Whenever a simulation object receives a message with time stamp less than a time stamp of a message already processed, a ‘rollback’ occurs. The rollback restores the simulation object’s state to the time stamp of the message that caused the rollback. Simply rolling back the process to a state at a previous time does nothing to nullify cascaded messages that have been sent out by the process. For example a simulation object at time $t + 10$ receives a message that causes it to rollback to time t. All of the messages that the object passed in the 10 time cycles that were rolled back must be nullified. Jefferson’s Time Warp elegantly handles the situation by having the object that ‘rolled back’ send anti-messages to nullify messages which are not longer valid. Anti-messages cascade if necessary.

A subtle shortcoming of Jefferson’s Time Warp is that it does not define the *event horizon*– how far ahead the simulation should process messages. An illustration of this shortcoming is evident from a scenario in which events are processed optimistically very far ahead and then a straggler message causes an explosive amount of cascading rollbacks. If such events happen too often, the simulation’s performance will suffer because communications channels can be saturated with anti-messages.

The Breathing Time Buckets technique addresses the problem encountered by Jefferson’s Time Warp by defining an event horizon as some time $t_{horizon}$ such that $t_{horizon} = t + \Delta t_{bucket}$. In other words, events are only optimistically processed up until a point at which the event horizon stalls and waits for the Global Virtual Time (GVT) of the simulation to reach $t_{horizon}$. Conceptually, the optimistic processing is ‘bucketed’ into groups. The advantage of Breathing Time Buckets is that excessive rollbacks do not occur. Still, on the other hand, the simulation cannot fully exploit the concurrency available to it.

Jefferson's Time Warp is computationally risky, while Breathing Time Buckets can be too conservative. Breathing Time Warp attempts to address the opposing characteristics of Jefferson's Time Warp and Breathing Time Buckets by combining the two into a new synchronization process that hallmarks each of their strengths [44]. In Breathing Time Warp, events are initially processed optimistically with some level of risk N_{risk}. Once the number of optimistically processed (but uncommitted) events get beyond N_{risk}, Breathing Time Buckets takes effect until the GVT is updated such that more events can be optimistically processed. Clearly, defining the parameter N_{risk} is critical to successful operation of Breathing Time Warp.

Corner's M.S. Thesis research applied BTW via the Synchronous Environment for Emulation and Discrete Event Simulation (SPEEDES) libraries to the AFITSS but with irregular results and ultimately no speedup occurring. Chapter 6 contains analysis the specifically compares conservative and optimistic techniques with regard to that particular application.

2.3.5 Parallel Computing Metrics.

2.3.5.1 Amdahl's, Gustafson-Barsis's, and Sun-Ni's Laws. At the heart of parallel computing is the ratio of communication time to calculation time ($\frac{t_{comm}}{t_{calc}}$). This ratio describes how much processing occurs on a CPU compared to the time it takes to communicate between CPUs in a parallel execution. Three 'laws' in particular are of interest with regard to this ratio: Amdahl's Law, the Gustafson-Barsis's Law, and Sun-Ni's Law.

Amdahl's Law makes the case for fixed size speedup. This conjecture states that speedup saturates and efficiency drops as a consequence of holding the problem size constant and increasing the number of processors. In other words, it implies that diminishing returns occur for speedup simply by increasing the number of processors. The equation for Amdahl's Law is given by Equation

2.3, where P represents the parallel processing time, and S represents the best possible serial processing time. For this particular application, analysis with Amdahl's Law is straight forward. To summarize, various problems of fixed size N (number of UAVs in the swarm) can be tested for increasing numbers of increasing processors. More thorough analysis of Amdahl's Law is found in [33].

$$Speedup = \frac{1}{1 - P + \frac{P}{S}} \tag{2.3}$$

Gustafson-Barsis's Law circumvents Amdahl's Law by increasing the problem size as the number of processors increase. It exploits the ratio between the portion of serial code in the problem and the problem size. An excellent explanation of this law is given by Thi*é*baut [66]. Essentially, it requires carefully decomposing the parallelization such that the serial portion and the parallel portion of the problem to a value of one unit of time–scaling up the problem to fit the machine. This conjecture too can be applied to the current application of this research by increasing the problem size N as the number of processors increases. In this formulation, R_s represents the portion of code that is serial.

$$Speedup = N - (N - 1)R_s \tag{2.4}$$

Sun-Ni's Law is a generalization of the previous two conjectures. It states that making all data accessible by all processors, using all available memory space, and increasing memory space as needed results in nearly ideal speedup [60]. Without a shared memory system (which would be ideal for this work), this metric may not be possible to exploit.

2.3.5.2 Speedup, Efficiency, and Isoefficiency. Other relevant metrics for any PDES include speedup, efficiency, isoefficiency, and scalability. These metrics give insight into the theoretical improvements that are gained by parallelizing a serial algorithm. It is not necessarily the case that parallelizing an algorithm results in improvement. It is certainly not the case that parallelizing beyond an additional number of processors produces improvement. These metrics algorithm the various ‘laws’ described provide the theoretical basis of what one can expect when parallelizing an algorithm.

Speedup is given by Equation 2.5 and can be measured by comparing serial runs to parallel runs of the simulation. In general, theoretical analysis with Amdahl’s Law provides the baseline with which to start analysis. In most cases, it is possible to achieve speedup beyond that which Amdahl’s Law provides by using techniques gleaned from Gustafson-Barsis and Sun-Ni’s research. Calculating speedup can be as simple as recording the wallclock time to run an experiment on a series of two or more processors and calculating the ratios of these times to the serial run. Speedup eventually plateaus and then begins to decrease when communication inefficiencies outweigh the additional calculation from the additional processors.

$$Speedup = \frac{T_s}{T_p} \tag{2.5}$$

The efficiency equation is closely related to speedup equation and conceptually describes how much each processor in a parallel operation is being used. Equation 2.6 shows that efficiency (E) can be defined as the ratio between the serial implementation in comparison to the parallel implementation multiplied by the number of processors used in the parallel implementation. The measure of efficiency is very much dependent upon the problem decomposition.

$$E = \frac{T_s}{pT_p} \tag{2.6}$$

Isoefficiency provides a metric that relates how to keep the efficiency of the simulation fixed while increasing the number of processors. The standard expression for isoefficiency is given in Equation 2.7, where W is the problem size, T_o is an overhead function, and K is a constant depending on the efficiency to be maintained. In short, isoefficiency expresses the relationships between problem size, an overhead function, and the number of processing elements. It is desirable to keep the efficiency fixed as the number of processors increases [33].

$$W = KT_o(W, p) \tag{2.7}$$

2.4 Summary

This chapter provided the pertinent background information relevant to routing swarms of UAVs. Because the problem of routing large swarms of UAVs can be instantiated as a VRP, an overview of the VRP along with its related variants and benchmarks is presented along with historical approaches to solving the problem. A particular emphasis on evolutionary computational techniques relevant to routing swarms of UAVs is mentioned to include Genetic Algorithms, Ant Colony Optimization, and Artificial Immune Systems. Background on swarm dynamics and parallel simulation is also presented. An understanding of these concepts provide the basis for the following two chapters, which present the modeling process for routing swarms.

III. Conceptual Design and Modeling for UAV Swarm Routing

If in other sciences we should arrive at certainty
without doubt and truth without error, it behooves us
to place the foundations of our knowledge in mathematics.
–Roger Bacon

Effective design and efficient modeling are prerequisites to the computational fruition of any scientific endeavor. No meaningful model can exist apart from a sound design. Desirably, a non-functional requirements specification identifies the specific scope and features of a project, and a sound design produces a detailed conceptual model that satisfies the requirements specification. This chapter outlines some generic design and modeling concepts and applies these concepts to the routing problem of large swarms of UAVs. Additional issues such as how to embed particular problem constraints into the model and routing algorithm are also addressed.

3.1 Overview of Design and Modeling

To design something is to conceive or systematically fashion it in the mind. Design is a creative and non-algorithmic process, and a good one produces a precise model [10]. A model is an encapsulation of some slice of the real world within the confines of the relationships constituting a formal mathematical system [13]. Because any given element of reality cannot ever be modeled, various constraints are an essential part of the formal system because they define the boundaries for which the model holds. A 'good' model characterizes a system with a desired degree of accuracy, and thus, enables accurate predictions of the natural system in consideration to be made.

Micro-UAV design is particularly interesting because they require rather unique design characteristics and must succumb to the laws of physics to high degree of fidelity. Available technology naturally influences these design considerations; the physical size of available technological devices, construction costs (especially if custom hardware is necessary), and the impact of these individual components on the overall aerodynamics of a UAV are just a few of the elements to analyze. For example, a requirements specification for a UAV might require that it stay airborne and continually collect sensor data for many days at a time. A design consideration must then address whether or not a custom built ultra-dense storage device for a UAV is within budget. If it is within budget, it must be able to fit in the hull of the UAV and have a benign effect on the aerodynamic design of the UAV.

3.2 UAV Simulation

A simulation is much like a black box in the sense that from a particular vantage point, some portion of reality is modeled and behaves as the model's design indicates that it should[1]. An adequate UAV simulation must consider several components in addition to the actual UAVs themselves, and there are several ways to decompose the space. A straight-forward approach to identifying the components produces at least the following items of interest:

- individual UAV characteristics
- swarm topology and size
- communication protocol and its scalability
- routing algorithm
- constraints from physics
- grid-mapping and spatial characteristics

A high fidelity near real-time simulation, efficiency is crucial, and thus, each of these elements have at least some degree of interdependence on the others. For example, modeling UAVs as point masses could ease the computational burden of calculating updates to position, but the effects of

[1]Emulation should not be confused with simulation. Emulation is more of a white-box approach which produces the expected input-output mappings by actually mimicking the internal workings of a system. For example, a simulation of a storage disk might simply return a byte array for a particular sector of a disk when prompted in a black box approach. An emulation of a storage disk, however, would require some degree of mechanical modeling to include the internal spinning of the disk, the seeking of the read head, and the reading of a sector before returning a byte array with the desired values

Table 3.1: Simulation Design Considerations

Hardware Platform	Software Platform	Development Tools
Num of CPUs	Threading	Systems Programming Languages
CPU Speed	Interprocess Communication	Scriping Programming Languages
CPU Memory	Existing Software Libraries	Development Environment
CPU Cache	CPU Scheduling	Debuggers
Amount of Persistent Storage	Memory Management	Versioning Management
Disk Read/Write Speed	Virtual Memory	Programming Language Documentation
Backplane between CPUs	Communications Interface	Compilation Tools
CPU Comm Topology	Deadlock Handling	
NIC Buffer Size	Security	

this simplification would be almost unnoticeable if an environment with myriad negligible forces and constraints act on the UAVs. It is interesting that the fidelity of the simulation need not depend on the most accurate model of reality as could be described; rather, the fidelity of the simulation depends on the accuracy of reality proportional to the level of design.

3.2.1 Hardware-Software Interaction. Another acute facet of a UAV simulation is the interaction between hardware and software interaction. In a large-scale UAV simulation, a system designer must consider at least the following items in Table 3.1 that comprise the hardware and software mappings. The physical limitations of hardware ultimately pose the most overbearing constraints on a real-system, and a simulation should account for these constraints. For example, it would be unreasonable to simulate a production level micro-UAV that passively gathers infrared data as the size of a bumblebee if the smallest infrared sensor currently produced is several orders of magnitude larger than a bumblebee.

For reconnaissance UAV designs, persistent data management is a significant design factor that influences both the physical characteristics of a UAV as well as security considerations. Data that is passively collected and stored on a UAV can simplify security considerations because it eliminates the need for a broadcast, which might be detectable by hostile forces. Persistent storage if hefty enough, however, affects the physical characteristics of UAVs because the physical hull of the UAV must house the storage mechanism, and the UAV must maintain particular flight and maneuverability characteristics.

3.2.2 Software Model of the AFIT Swarm Simulator. The existing AFITSS is the logical fruition of Kadrovach's swarm model with parallel extensions by Corner [16,39]. Kadrovach's work produced an implementation of an unscalable swarm model that uses a global data structure and is designed for a serial processor. A benefit of this model is that it does accurately simulate the flocking behavior of a swarm and can be used in applications that require analysis of sensor coverage within the swarm. Disadvantages of the model include its unscalability because of the reliance on global communication within the swarm and its unroutability (currently, the swarm wanders in a pseudo-random fashion).

Corner's work [16] was not able to produce any speedup using optimistic event processing with the Breathing Time Warp algorithm, and further investigation on this topic is the focus of part of the experimentation and analysis as described in Chapters 5 and 6. Recent work from Rome Labs shows that custom hardware and modification to the SPEEDES libraries can further improve performance for some applications [3].

A conceptual architecture of the existing simulator is presented in Figure 3.1. The existing simulation comes in two varieties: 'DIScline' and 'pswarm'. The former is a Linux port of Kadrovach's Visual C++ implementation, while the latter is a parallelization of the Linux port by Corner. The remainder of this section discusses the existing software architecture of the AFITSS.

3.2.3 SkyView. SkyView is a visualization tool originally developed by Ball Inc. The Air Force Research Labs Virtual Combat Laboratory (AFRL/SNZW) has a licensed copy with permissions to alter and maintain it for specific AF applications. SkyView can visualize any simulation adhering to the IEEE DIS standard [38], and provides a bird's eye view of the battle space. Individual actors, including man-in-the-loop actors, can operate at their own individual terminals, while commanders can view the entire battle space or specific portions thereof. Figure 3.2 shows a screenshot from AFRL/SNZW's Skyview. SkyView's user interface includes a help menu, extensive zooming options, fully spherical viewing of the battle space, and loads real Digital Terrain Elevation

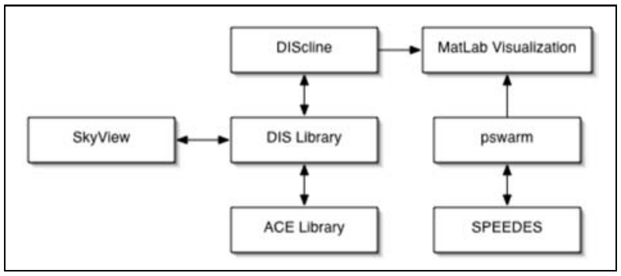


Figure 3.1: The overlying conceptual architecture of the simulation showing which components interface directly with other components.

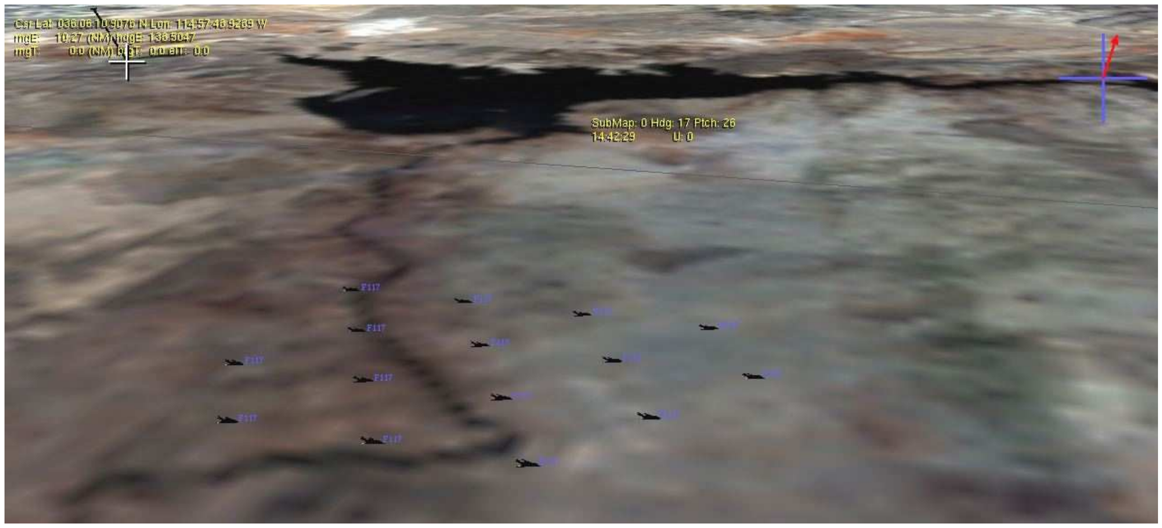


Figure 3.2: A ‘kswarm’ of UAVs flying in Skyview.

Data (DTED) that can be configured. Visualization is of particular importance for swarm research because it is saves much time and tedious analysis. Instead of analyzing sequences of matrices and output files filled with numbers, a visualization system provides a manner of actually seeing the results. Skyview, in particular, allows for viewing in near real-time, which is a critical aspect, because future work will undoubtedly become more interactive. The ‘newlibdis’ communications library interfaces to Skyview by means of a DIS daemon. Currently, SkyView is a one-way pipe capable of accepting data but not passing any data back. Primitive data passing in the opposite direction is necessary if real time user interaction is to be accomplished using SkyView.

3.2.4 SPEEDES.

SPEEDES (Synchronous Parallel Environment for Emulation and Discrete Event Simulation) is a freely available open source research initiative that has been maintained by the Metron High Performance Computing (HPC) group for over a decade[2]. It is a very flexible, robust, and well documented system intended specifically for modeling parallel discrete event simulations. For these reasons, it was selected as the basic framework for Corner's pswarm [16]. SPEEDES is written in C++ and is an Object Oriented (OO) library. This is very convenient and appropriate for discrete event simulations because of the inherent OO nature of simulation itself. In particular, this allows each particle of swarms to encapsulate its directional velocities and heading angle, which eases software development. Message Passing Interface (MPI) support is not available for SPEEDES and is not expected to become an initiative, because over the redundant overhead that would be imposed when marshaling data to and from nodes. In fact, recent efforts at Rome Labs have endeavored to remove much of the unnecessary communication in the TCP stack in lieu of a more lightweight Myrinet implementation, improve algorithms in SPEEDES, and incorporate specialized hardware [3]. Preliminary results show that optimizations to the SPEEDES library can improve performance, but definitive results on the topic are still pending.

3.2.5 ACE Libraries.

The ADAPTIVE Communication Environment (ACE) is a freely available, open-source OO framework that implements many core patterns for concurrent communication software[3]. ACE provides a rich set of reusable C++ wrapper facades and framework components that perform common communication software tasks across a range of OS platforms. The communication software tasks provided by ACE include event demultiplexing and event handler dispatching, signal handling, service initialization, interprocess communication, shared memory

[2]The SPEEDES User's Guide [44] provides a very comprehensive introduction to the framework and presents the system in such a way as to ease the reader into the system in small increments. Complete source code for each example in the manual is available at `http://www.speedes.com`. One need only have a GCC compiler to use the code. Each example is designed to illustrate an important capability of the system. Theoretically, working through the entire manual and having a comfortable background in the C++ programming language is all one should need in order to effectively use and understand SPEEDES.

[3]This introductory paragraph taken from [56]

management, message routing, dynamic (re)configuration of distributed services, concurrent execution and synchronization.

The ACE libraries provide the underlying communication infrastructure upon which the 'newlibdis' library is implemented. In many cases, DIS functions are nothing more than an adaptive wrapper around an ACE function. A general understanding of how the ACE libraries work is important for understanding some of the underlying technical aspects of AFITSS, but directly interfacing to ACE should continue to be channeled through the DIS interface wrappers.

3.2.6 'newlibdis' DIS Library. The 'newlibdis' DIS library is a development of AFRL/SNZW, built upon the ACE libraries, and provides a one-way communication pipe to Sky View. Essentially, 'newlibdis' uses foundational C++ objects to construct Protocol Data Units (PDUs) that relate information about the entity state of an object in the real world such as a UAV or radar site. 'newlibdis' uses extrapolation (dead reckoning) to tract the position and velocities of objects in order to conserve network bandwidth. While some additional features such as multicast capability and expanded PDU support need to be added to the library, 'newlibdis' serves the AFRL VCL as a specific implementation of the DIS standard that allows multiple DIS applications to run on a single machine.

3.2.7 Kadrovach's kswarm. Kadrovach developed a system for modeling swarm based communications [39]. A primary result of his Ph.D research was 'kswarm', an implementation of a serial algorithm having $O(n^4)$ complexity that simulates a swarm of particles. His experiments and work are very well documented in his dissertation and improving his work was very much Corner's primary effort and continues to be no small part of current AFIT research. A primary issue pointed out by Corner [16] is the need for a global controller in 'kswarm' and that the serial communications model is not entirely accurate.

The serial communications model inaccuracies are illustrated via the given scenario: Given a swarm of sensors S, suppose that $s_i \in S$ is updated at t=1. Sensor s_j makes a movement based on s_i's *new* position, Kadrovach's algorithm maintains that sensors s_i and s_j moved at exactly the same time. Corner addressed this change, although it resulted in little difference for the flocking behavior of the swarm, considerably smoother swarming behavior emerged [16]. 'kswarm' as produced by Kadrovach is written in Visual C++, and so it has dependencies on the Microsoft Foundational Class (MFC) libraries. Corner replaced these dependencies with their QT equivalents with the port to Linux. Visualization of 'kswarm' scenarios is accomplished via watching 'kswarm' binary output files as MatLab movies.

3.2.8 Corner's 'DIScline' and 'pswarm'. Before parallelizing 'kswarm', Corner accomplished a serial port to Linux, 'DIScline', and was able to visualize via SkyView through calls to the 'newlibdis' API. A parallelization of the serial port resulted in 'pswarm'. Parallel simulation of 'pswarm' is accomplished with AFIT's Beowulf cluster. Corner then parallelized the work and dubbed the parallelized version 'pswarm'. 'pswarm' is a straight forward parallelization of Kadrovach's algorithm in that it uses continues to use Kadrovach's global data structure–a giant bottleneck in terms of scalability. 'pswarm' uses SPEEDES for a scheduling mechanism, and with minor exceptions, SPEEDES is the "ideal" software library for implementing a PDES. 'pswarm' improves the 'kswarm' algorithm by lowering it's algorithmic complexity to $O(n^3)$ through eliminating an outer loop with the parallelization. Although 'pswarm' leaves much to be sought, it does provide proof-of-concept for a more high fidelity simulation and addresses many of the considerations necessary for developing a more advanced parallel swarm simulation.

3.2.9 Running the AFIT Swarm Simulator and Hardware Configurations. Corner documents the extensive use of scripts in order to accomplish the many tasks involved with running a parallel SPEEDES application using a Portable Batch Scheduling (PBS) system. As of his research, the configuration of the nodes in the Beowulf cluster involved only 32-bit nodes, with the ability to use 1 or 2 processors per node (sharing memory) with either a fast ethernet or Myrinet backplane. Nodes 1-32 in the aspen “workq” use the fast ethernet backplane, while nodes 33-48 in the aspen “myriq” use the faster Myrinet backplane. Although the simulation could theoretically run in the “polyq” having a fast ethernet backplane with very minor changes to the scripts, the “polyq” is not meant for parallel processing per AFIT computing protocol.

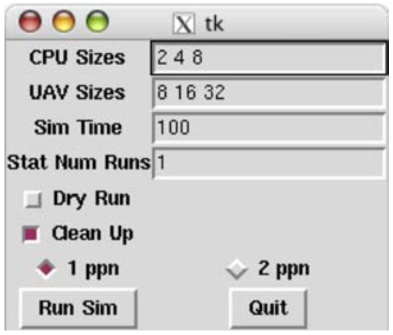
Since Corner’s research some hardware changes have taken place. The most notable changes to hardware that could potentially impact the parallel simulation include 1) there is only one processor per node available on the aspen “workq” and 2) a new 64-bit cluster (tahoe) was installed.

Because the scripts were in an unmaintainable state, many of them were rewritten for readability, understanding, and maintenance. Many other artifacts in their containing directory were eliminated or categorized into subfolders for similar purposes. Of notable interest, the “start.sh”, “doRuns.sh”, and “runMatrix.sh” scripts were rewritten. Because the newer scripts are built such that they rely on environment variables instead of hardcoded paths, some elementary configuration is necessary by setting a few paths in a users Bash profile. Scripts were rewritten with the ability to operate on the 64-bit cluster as well, so variables need to point to different compilations depending on which head node the user is logged into. The actual SPEEDES library compiled successfully for 64-bit processors but complications with output from the “external module” prevented data collection and analysis.

An additional burden to running the simulation involved passing a long string of arguments to “start.sh” with each run of the simulation. In order to provide a more friendly interface, a simple Python GUI frontend was built that facilitates the process and is show in Figure 3.3.

3.3 Shortcomings of the Existing Swarm Simulator as a SPEEDES Application

The primary issue with the current implementation of Kadrovach’s work as a SPEEDES application as implemented by Corner is two-fold: 1) the algorithm’s use of a global data structure


tk

CPU Sizes: 2 4 8

UAV Sizes: 8 16 32

Sim Time: 100

Stat Num Runs: 1

☐ Dry Run

[✓ Clean Up]

[✓ 1 ppn] 2 ppn

Figure 3.3: A simple GUI interface for running the AFIT Swarm Simulator. Previous usage was tedious and involved passing all arguments to the application via the command line.

and 2) the enormous number of rollbacks involved in the Breathing Time Warp optimistic processing algorithm. Although the algorithm's running time is polynomial, Corner's analysis of his own implementation shows that it does not scale for swarm sizes larger than 100. Ideally, AFITSS should be able to simulate many thousands of micro-UAVs. Corner's research measured varying configurations of UAVs in simulation for serial and parallel runs. Corner was surprised to discover that the parallel implementation slowed to a crawl for an experiment involving only 500 UAVs with 10 processors using the SPEEDES library. Corner's particular experiments eventually concluded that "the working parallel swarm SPEEDES implementation was not efficient enough for running larger UAV counts (> 100)" [16]. More in depth analysis of the simulation performance is presented.

The science fiction work *Prey* [19] provides an interesting and not entirely unbelievable scenario involving very large swarm sizes. The primary factor driving the scalability and efficiency issues with the current algorithm is the need for each member of the swarm to communicate with $n-1$ other members of the swarm at each time step. Therefore, adding an additional swarm member introduces $n-1$ additional communications for each original member of the swarm. HPC hardware and implementation can only provide so much relief. In short, the current version of 'kswarm' and its underlying data structure simulates swarm behavior for small swarm sizes but cannot simulate very large swarm sizes; 'kswarm' does not address the routing of swarms.

For large scale simulation, future research efforts must produce a new and scalable communications algorithm. This algorithm should inherently integrate other considerations such as routing as applicable. It is not reasonable to develop a swarm algorithm that does not have an routing mechanism embedded and "attach" this capability to the algorithm after its inception. Rather, the routing aspect of the algorithm should be part of the swarm algorithm model and abstractly verified before the actual implementation begins for the simulation. To state it another way, the routing algorithm inherently has a tight coupling to the internal behavior of the swarm. A swarm that is capable of being routed from one coordinate to another must have an underlying model that addresses at least these two central issues.

3.4 Conceptual Model for a Swarm of Particles

In order to model a swarm, each member can be viewed as a particle. The characteristics of a single UAV are encapsulated as a tuple and then a swarm of k UAVS is simply a set of these tuples. A sequence is necessary to model the swarm because each UAV must be uniquely identified.

The essential characteristics of a UAV include the relative X, Y, and Z coordinates, X, Y, and Z velocities, a heading angle relative to an offset, and a max turn angle. In general, the offset angle of the heading may be uniform for the entire swarm but is provided here for completeness. Other important aspects of UAVs to consider include possibly heterogeneous fuel or energy capacities, weights of the UAVS, and loading capacities.

$$u = \{x, y, z, v_x, v_y, v_z, \theta_{offset}, \theta_{heading}, \theta_{max-turn}, C_{fuel}, Weight, C_{loading}\}$$

If the UAVs are considered point masses, these parameters suffice, but if the UAVs are to be modeled as realistically as possible, then it may be appropriate to also consider their potentially heterogeneous wingspans, hull lengths, and girths. The parameters roll, pitch, and yaw may also be considered at a much lower level design. Based upon a tuple designating a UAV, the swarm then is simply a *sequence* of UAVs.

$S = u^k = \{\vec{u}_1, ..., \vec{u}_n\}$

3.5 Problem Constraints and Parameters

Essential problem parameters and constraints for routing a swarm of UAVs in a simulation include, but are not limited to:

- A minimum or maximum number of UAVs in the swarm:
  - $|S| = n$
- Starting and ending coordinates of the swarm, where a point is $P = \{x, y, z\}$:
  - $\{p_{start}, p_{end}\}$
- A boolean value indicating whether or not to self-destruct in the case of a fatal event:
  - $SD = \{0, 1\}$
- The notion of a neighborhood and/or visibility within the swarm
  - $\forall u \in S \exists \{\vec{u}_1, ..., \vec{u}_n\}$
- A sequence of way points, sorted by priority:
  - $W = P^k = \{\vec{p}_1, ..., \vec{p}_k\}$
- Time windows, TW, (hard or soft) associated with the way points and their penalties:
  - $\forall p \in W \exists t_{min}, t_{max}, C_{penalty}$
- A sequence of sorted threats:
  - $T = \{p, risk\} \times_k \{p, risk\}$, where $risk \in [0, 1] \wedge p \in P$
- A maximum mission duration based upon fuel and velocity constraints. For this work, consider $C_{fuel} \propto t_{max}$:
  - t_{max}
- A tolerance, C_ϵ to determine the proximity with which the swarm must come within way points:
  - $\forall p \in W, \exists r \in \Re$
- The minimum precision with which to conduct calculations:
  - 10^{-k}

3.6 Adapting the VRP to Swarming Reconnaissance

Previous work by Kadrovach and Corner provides a proof-of-concept for a working parallel swarm simulation and allows various metrics regarding parallel performance and swarm behavior to be collected and analyzed. Existing work does not, however, provide a means of routing the swarms from point to point–an essential characteristic for any type of mission planning system.

Fortunately, routing a swarm maps nicely to an instantiation of the hard problem VRPTW, and much research on this hard problem is published and readily available[4].

In a routing model, UAVs must travel through a theatre and deliver a particular amount of reconnaissance payload to a set of targets. Each target may have particular time windows, and the total 'capacity' of each UAV is proportional to its fuel supply[5]. Once an instantiation of VRPTW with additional threat constraints is solved, the swarm is subdivided according to waypoint priorities and their associated threat levels. The subswarms then route through each circuit of the routing sequence as a single vehicle, respond to a changing threat environment in their circuit independently, and behave otherwise independently of other subswarms. Low level behavior parameters and swarming characteristics for each subswarm may be tuned and set appropriately. Subswarms navigating areas of threat would be tuned to behave differently than subswarms navigating areas of relatively low threat depending upon mission criteria.

A generic mathematical instantiation of the swarming reconnaissance problem may be formulated as follows: There exists a set of waypoints W, each requiring a payload of reconnaissance known *a priori* c_k, such that $\forall w_i \in W \exists c_n$, where $n > 0$. A homogeneous tuple of UAVs $S = u^m$ exists and each $u \in S$ has fixed fuel capacity such that $\forall u \in SC_{fuel}(u) = C_{fuel}(u)$. There also exists a set of costs, $Q \ni q_{ij}$, incurred by traveling amongst any two way points w_i and w_j. A circuit G obtained from $W \times_k W$ is defined as a sequence from W that starts and ends at p_{start}. The solution to this instantiation of the VRP is the set of Z simple circuits given by $Z \ni min \sum_{g \in G} \sum_{q \in Q} q$ subject to the following constraints:

- No partial reconnaissance payloads are allowed.
  - $\forall w_m \exists c_i \ni c_i = c_{req}$
- All reconnaissance activity must be fulfilled.
  - $\forall l_i [\exists q_{ai}, q_{ib} \in Z]$

[4]Routing a swarm via a VRP solution presupposes the capability of a swarm to route from one coordinate to another coordinate as one of its inherent characteristics, preferably in 3 dimensions. The process of routing the swarm as referred to by a VRP solution is different from the swarm's ability to 'route' or traverse an edge on a graph.

[5]Assume travel distances between targets is negligible or account for the travel distances by subtracting an upper threshold from the UAVs maximum fuel supply

- The only intersection of the circuits in Z is p_{start}.
  - $\bigcap_{\forall z_i \in Z} = p_{start}$
- Each route must have length less than or equal to the maximum distance the swarm can travel.
  - $\forall u \in S, \sum q_{ij} \in Z_u \leq t_{max}$
- Time windows must be met for all way points.
  - $\sum_{tw \in TW} \cdot \delta_{tw} = 0$, where $\delta_{tw} = 0 \Leftrightarrow \forall w \in W \exists u \in S \ni At(u,p) \wedge t \leq tw_{max}(w) \wedge t \geq tw_{min}(w)$

All constraints except for the final two are standard CVRP constraints. The time windows constraint is the standard VRPTW addition to the CVRP, and the final constraint is necessary for modeling a hostile environment. The time windows constraint increases the combinatorics of the problem by orders of magnitude, and modeling threats in the environment only associates an additive cost to waypoints. Incorporating threats into the routing evaluation function is handled by associating a threat level with each waypoint instead of modeling the threats as waypoints in and of themselves. This methodology adds a measure of robustness by reasoning about the uncertainty associated with threats and keeps the routing algorithm simpler than it would be than by attempting to route around threats.

A necessary step toward refining a routing model that represents threats is to define the *waypoint neighborhood threat level* for any given waypoint k in terms of concentric equidistant rings of varying proximity c to the waypoint k.

Definition 3.1. *A waypoint k's neighborhood threat level $N_{i,j}(k)$ is proportional to the additive cost of all threats contained in the circular distance surrounding k with radius $i \leq r \leq j$, where $\forall i \forall j (j - i) = c$.*

The threat level associated with any given waypoint k is a function of the proximity of all of the threats in any given neighborhood of k with maximum j value j_{max}:

$$TL_k = \Sigma_{i=1}^{j_{max}-c} N_{i,i+c}(k) \Delta c \tag{3.1}$$

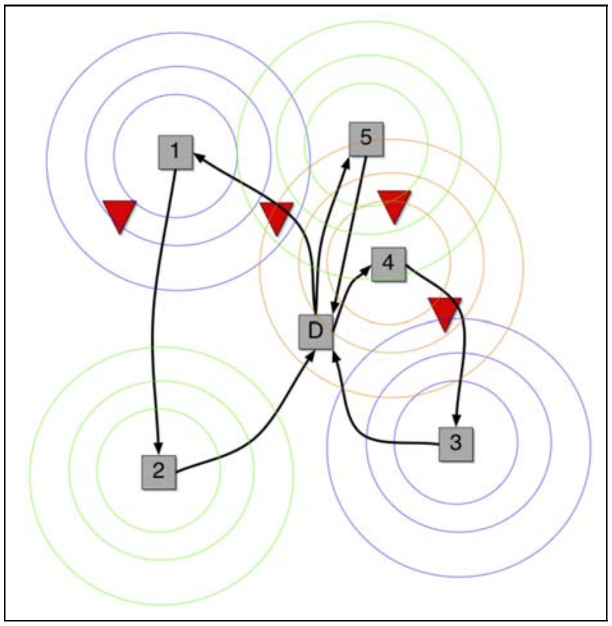


Figure 3.4: A routing scenario.

Figure 3.4 illustrates one such hypothetical routing scenario for this given model. The center of the figure is a central depot location where all subswarms begin and end their routes. Red triangles are threats, and numbered squares are waypoints. The cost of visiting a waypoint is proportional to the distance to travel to the waypoint and the threat level associated with it. Each waypoint's threat level is a calculation that takes into consideration the proximity of surrounding threats.

Modeling threats in this particular manner provides robustness for the model by not requiring precise information about any given threat. Rather, the uncertainty for any given threat may be represented in the neighborhood threat levels of the surrounding waypoints–resulting in a simpler routing algorithm. This outcome is desirable since the VRP is already a hard combinatorial problem. An additional property of modeling threats by varying degrees of certainty and uncertainty is that the routing algorithm is more resilient to changing threat levels than it would be by modeling threats as separate entities and routing UAVs around them in a 'cat-and-mouse' type game.

Given the aforementioned routing and swarm model, Algorithm 2 presents a high level controller for a mission planning system.

3.7 Summary

High-level design for a UAV simulation involving mission routing is prerequisite to a lower level modeling. The high level design presented in this chapter reviews a conceptual architecture of the existing AFIT Swarm Simulator and addresses some of the issues with the current implementation that must be overcome in order to produce a high fidelity Parallel Discrete Event Simulation capable of routing very large swarms of UAVs. In particular, scalability, software component architecture, and routing issues are addressed. This high-level design considers bottlenecks and efficiency problems between components of the model and also address effectiveness in the sense of the model being able to complete an assigned task as determined by a requirements specification.

Low level design investigates in detail the pertinent aspects of the high-level model in consideration and addresses specific efficiency issues of selected components of the model. The low level design for a specific approach to routing swarms of UAVs fills the bulk of the next chapter and is a cornerstone to a realistic high fidelity PDES capable of supporting Air Force organizations such as the Air Force Research Laboratories Electronic Warfare Sensors Directorate (AFRL/SNZW) Virtual Combat Laboratory (VCL).

```
Data   : $S$, $p_{start}$, $p_{end}$, $t_{max}$, $T$, $W$, $TW$, SD, $C_\epsilon$
Result : $Visited\_Waypoints$, $Waypoint\_Visitation\_Times$,
         $Waypoint\_Epsilon\_Values$, $SWARM@p_{end}$
$Routing\_Sequence := Calculate\_Routing\_Sequence(W, T, TW)$;
$Mission\_Duration := Calculate\_Duration(Routing\_Sequence)$;
$Max\_Duration := Calculate\_Max\_Duration(S)$;
if $Mission\_Duration \leq Max\_Duration$ then
    'Error: Mission Duration exceeds fuel constraints';
end
$SubSwarms := Subdivide\_Swarm(Routing\_Sequence, T, |W|)$;
/*In Parallel*/
for $\forall s \in Subswarms$ do
    $Waypoint\_Sequence := Calculate\_Waypoint\_Sequence(Routing\_Sequence)$;
    $Visited\_Waypoints := \emptyset$;
    $Waypoint\_Visitation\_Times := \emptyset$;
    $Waypoint\_Epsilon\_Values := \emptyset$;
    $t = 0$;
    while $t < \frac{Max\_Duration}{2}$ do
        for $\forall u \in s$ do
            $Calculate\_Update(u)$;
        end
        for $\forall u \in s$ do
            $Update(u)$;
        end
        $Next\_Waypoint := Head(Waypoint\_Sequence)$;
        $Waypoint\_Epsilon := Dist\_to\_Waypoint(Center\_Of\_Mass(s), Next\_Waypoint)$;
        if $Waypoint\_Epsilon < C_\epsilon$ then
            $Waypoint\_Visitation\_Times := Waypoint\_Visitation\_Times \bigcup t$;
            $Visited\_Waypoints := Visited\_Waypoints \bigcup Next\_Waypoint)$;
            $Waypoint\_Epsilon\_Values := Waypoint\_Epsilon\_Values \bigcup Waypoint\_Epsilon$;
            $Waypoint\_Sequence := Waypoint\_Sequence - Next\_Waypoint$;
        end
        Calculate_Threat_Update;
        if $Threat\_Update$ then
            $Routing\_Sequence := Calculate\_Routing\_Sequence(Waypoint\_Sequence, T, TW)$;
            $Waypoint\_Sequence := Calculate\_Waypoint\_Sequence(Routing\_Sequence)$;
        end
        if $Is\_Fatal\_Event \wedge SelfDestruct$ then
            Destroy_Swarm;
        end
        $t = t + \Delta t$
    end
    while $Waypoint\_Epsilon > C_\epsilon$ do
        Calculate_Threat_Update;
        $Routing\_Sequence := Go\_Home(p_{end}, T)$;
        $Waypoint\_Epsilon := Dist\_to\_Waypoint(Center\_Of\_Mass(s), p_{end})$;
        for $\forall u \in s$ do
            $Calculate\_Update(u)$;
        end
        for $\forall u \in SWARM$ do
            $Update(u)$;
        end
        if $Is\_Fatal\_Event \wedge SelfDestruct$ then
            Destroy_Swarm;
        end
    end
end
Land_Swarm;
```

Algorithm 2: Pseudo-code for high level swarm control.

IV. Low Level Design for UAV Routing

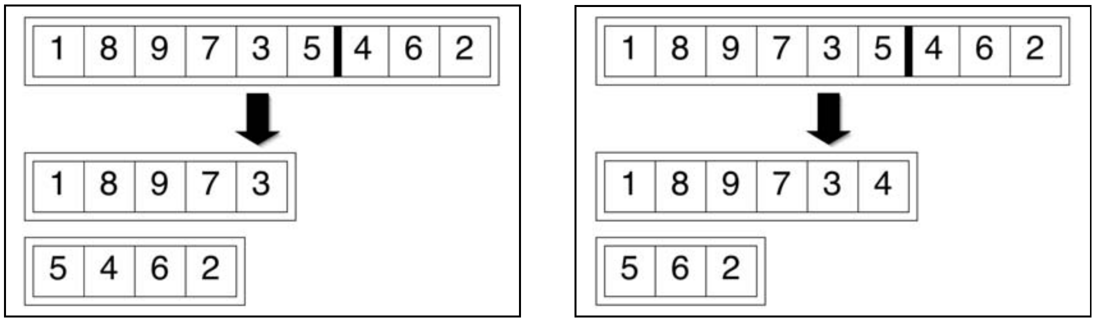
Nothing in life is to be feared.
It is only to be understood.
Marie Curie

THE previous chapter provided background on a specific implementation of a swarm model and a parallelization of this model. It also introduced a high level model for routing swarms of UAVs. This chapter extends that base with a detailed discussion of a particular routing algorithm implemented using a Genetic Algorithm (GA). Exploration and exploitation characteristics of a particular GA implementation are presented as a mathematical model, and particular techniques for embedding problem constraints are investigated. The next chapter presents experiments for measuring efficiency of this algorithm using well-known benchmark problems, an in depth landscape analysis for the problem domain, and parallel simulation analysis.

4.1 Decoding Path Based Representation Solutions

A naive method for representing VRP solutions is with path-based representation (PBR). In PBR, henceforth referred to as PBR-1, a solution to the VRP is simply a single sequence of locations. In such a sequence, routes are obtained by walking the sequence from start to finish in order to group consecutive locations into groups. When adding the next consecutive location would exceed the total capacity of a vehicle, the current route terminates and a new route begins. This particular decoding procedure takes $\Theta(n)$ time for n locations. Figure 1(a) illustrates PBR-1.

A variant of this particular PBR method, designated as PBR-2, shown in Figure 1(b), involves a final attempt to more tightly pack the current solution. Instead of immediately terminating the current route and starting a new one, any possible location that could be added to the current solution from the remaining locations is added. This particular procedure in its worst case could potentially take $O(n^2)$ time. This worst case would involve each vehicle only being capable of serving a single location, and consequently, all possible attempts at adding an additional location



(a) PBR-1 (b) PBR-2

Figure 4.1: The decoding of a solution encoded with PBR. Assume that even numbered locations have a demand of one unit, odd numbered locations have a demand of two units, and vehicles have a capacity of ten units.

are in vain. In any real world problem, the worse case is not expected; if the worse case were to occur, an optimal solution would be trivial: to use a single vehicle for each delivery location. Still, a worst case time complexity calculation gives an upper bound on the decoding time. The time complexity for this worst case is trivial to prove with mathematical induction.

The best case for PBR-2 is given when the sequence of locations optimally packs the vehicles. In this case, the decoding sequence takes only $O(n)$ time.

4.1.1 Crossover and Mutation Operators. Any crossover or mutation operators compatible with the TSP are compatible with PBR for the VRP, since any feasible PBR solution can be interpreted to represent a Hamiltonian circuit for the problem. The most common PBR crossovers are partially mapped (PMX), order (OX), and cycle (CX) crossovers [45]. While an exposition of these operators is not pertinent to discussion, a brief summary of their behavior is in order.

The PMX operator is well-known and selects subtours of equal length from two parent solutions, exchanges them, and attempts to preserve as many of the remaining locations as possible. Locations where 'conflicts' would occur are repaired by a set of mappings given by the subtour exchanges. This operator exploits the similarities in the value and ordering between two solutions [32]. A one parent "remove-and-replace operator, however produced better results on an ordinal representation for a specific 100 city TSP problem [26].

The OX operator attempts to exploit the notion that the relative order of the cities in a solution are important [23]. For example, various permutations of locations may be identical solutions if the two are in fact the same Hamiltonian cycles. No known competitive results for the TSP were found in the literature for this operator.

The CX operator emphasizes the preservation of the absolute position of elements from the parent solutions [49]. The operator ensures that each element's location in the offspring solution comes from one of the parent solutions. Often this means that an initial selection of one gene from one parent is all that is needed in order to generate the rest of the entire solution. Which gene is selected is paramount to this operator's success.

Any operator that preserves a mapping to a Hamiltonian circuit is a candidate mutation operator. Common mutations may include swaps, displacements, and inversions [45].

4.2 Genetic Vehicle Representation

GVR [43, 51, 62, 63] provides the base for an effective routing algorithm. The forte of GVR is its data representation; its novel aspects include its crossover and mutator operators. A deeper analysis of these operators than Tavares presents is paramount to developing metrics for measuring GVR's efficiency and understanding its balance between exploration and exploitation.

GVR differs from PBR in that it *c*an explicitly provide the number of routes and locations in them for a given solution without significant decoding. For any given solution, there exist a set of routes where each route is a sequence of locations. In previous work by Tavares, each route may or may not meet feasibility criteria. If a route fails to meet feasibility criteria, it is implicitly repaired. That is, the solution is repaired on a meta-level and evaluated, but the actual building blocks of the solution remain unaltered. A Lamarckian model would explicitly repair the solution, evaluate it, and then replace the infeasible solution with the repaired solution in the population. Given these characteristics, the best case time complexity to decode a solution is $\Theta(n)$ for the case in which solutions are explicitly repaired and exactly n locations are visited with each decoding. The worst case in which no repairs take place and a mechanism such as PBR-2 is used to decode the solution is $O(n^2)$.

4.2.1 Crossover. The crossover operator is conceptually simple. Two individuals, C_1 and C_2 are chosen for selection. A subroute, r from individual C_1 is inserted into a copy of individual C_2 in such a way that the distance between the insertion location and the first location in r_1 is minimized. Mathematically this sentence is formulated as follows: $\exists k \forall l dist(k, r_1) < dist(l, r_1) \wedge k \neq r_1 \wedge k, r_1 \in L$. In general, guaranteeing a solution to the given sentence requires $\Theta(n)$ time for n locations in a solution since all possible solutions must be examined. For a given subroute r and donor S of size n, there exist n possible crossovers.

Clever techniques such as table lookups or hashes may decrease this time to a constant, but even then, it may still take $O(n)$ time to traverse the routes and insert r. Depending upon the particular implementation, an additional preprocessing time of $O(n)$ may be necessary in order to remove the duplicate locations from the surrogate such that once r is inserted, the solution is again feasible. Figure 4.2 illustrates the GVR crossover. The final step illustrating the repair operation occurs with some probability $1 \leq p \leq 0$. Note that the crossover operator can never create additional routes, because there always exists a location in an existing route that has a minimal distance to r_1. The crossover operator, however, can eliminate an existing route. This happens only when the subroute r already exists in the donor as its own route.

4.2.2 Swap Mutation. In swap mutation, two locations l_1 and l_2 in solution S are chosen at random and swapped with one another. These two locations may or may not exist in the same route, and there exists the possibility that both random locations may be the same (in which case

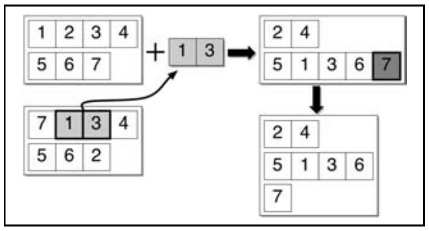


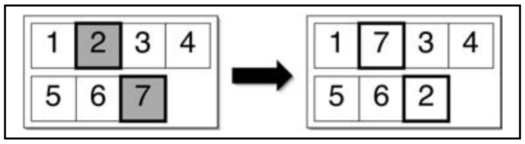
Figure 4.2: The crossover in GVR. A surrogate accepts an insertion of genetic material from a donor and a repair operator removes the original duplicate locations in the surrogate. The final step illustrating the repair operation occurs with some probability.

no change occurs as the result of the mutation). Thus, for a solution containing n locations, there are n possible swap mutations that may take place.

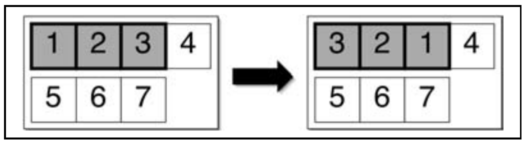
Notable properties of the swap mutator are that the number of locations in the affected routes does not change: $\forall R_{pre-swap} \forall R_{post-swap} \in S |R_{pre-swap}| = |R_{post-swap}|$. The length of the routes and total demand for the route, however, almost certainly changes. Additionally, swapping may result in a feasibility violation by causing a route to exceed the maximum capacity of a vehicle. The potential disruption for the swap mutator is comparatively low because at most two locations are affected. Consequently, even if frequent swaps occur in a model with low repair probability, the overall characteristics of the solution remain relatively similar for the general case.

4.2.3 Inversion Mutation. For inversion mutation, a subroute from a solution $r \subseteq S$ is chosen and inverted. For a solution containing n locations, there are $\Sigma_{i=1}^{n} \binom{n}{i}$ possible inversions including the trivial ones of which a subroute of size 1 is inverted and no actual mutation occurs.

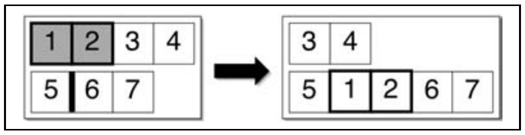
Notable properties of the inversion mutator are that the number of locations and total demand of the affected route does not change, $\forall R_{pre-inv} \forall R_{post-inv} \in S |R_{pre-inv}| = |R_{post-inv}| \wedge demand(R_{pre-inv}) = demand(R_{post-inv})$, and thus, feasibility criteria is not affected as a result of the mutation. The only change that may occur is the total distance of the route. Inversion mutation, however, can be very disruptive, *especially in models with very low repair probability*, and is not usually employed as a GA operator. The reason for this disruption is because the inversion



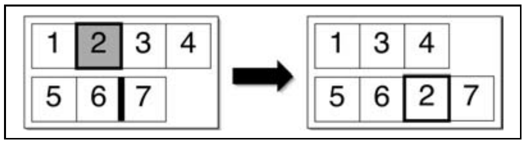
(a) Swap



(b) Inversion



(c) Displacement



(d) Insertion

Figure 4.3: GVR mutation operators

mutator can change a subroute of a solution in which the subroute is actually all of or parts of two separate routes.

4.2.4 Displacement Mutation. The displacement mutator selects a subroute from a solution $r \subseteq S$ and relocates it to another place which may or may not be in the same route. In fact, it is identical to the crossover operator except that it randomly inserts the subroute r instead of inserting it in such a way that the total distance after the operation is minimized. This random insertion point may result in the creation of a new route with some probability; Tavares uses the probability $\frac{1}{2V}$, where V is the number of vehicles in the current solution. Thus, the higher the number of routes in a solution, the lower the probability that an additional route is inserted.

Like the crossover operator, for a solution containing n locations, there exist $\Sigma_{i=1}^{n+1} i$ total possible displacement mutations including the trivial cases in which no change occurs. For a given subroute r of cardinality k, there exist $(n + 2) - k$ possible displacements in the solution. The possible disruption associated with this mutator is expected to be relatively high since possibly long subroutes may be displaced.

4.2.5 Insertion Mutation.

Insertion mutation is a special case of displacement mutation in which the subroute size of displacement is of size one. Thus, for a solution containing n locations, there are $(n + 2) - 1 = (n + 1)$ possible insertion mutations including the trivial case in which no change occurs. Potential disruption from insertion mutation is expected to be very low since at most one route and one location is affected. This holds true even in models with low repair probability in which very long infeasible routes may exist in a solution.

4.3 Embedding Specific Constraints into a UAV Routing Mission

4.3.1 Embedding Constraints in the Problem Statement.

Specific constraints to a UAV routing mission are part of the specific problem domain and should be incorporated into the problem statement. Consider the following minimal constraints for a UAV swarming reconnaissance mission:

- The sum of the reconnaissance for a route may not exceed a subswarm's capacity
- Time window constraints may not be violated for any way point in any swarm's route
- Threats known *a priori* must be evaluated and incorporated into a routing algorithm's objective function at a global level.
- Threats appearing dynamically while subswarms are en route shall be handled at the subswarm level.

Meeting the constraint that the sum of the reconnaissance for a route may not exceed a subswarm's capacity is typical of any VRP. In traditional delivery truck based approaches, this constraint required that the sum of the demands for each location in a route be less than the capacity of a delivery truck. With respect to UAVs, however, this constraint implies that the maximum time the subswarm can stay aloft must be greater than the required time to survey all of the waypoints in the route assigned to the subswarm. The time to survey is the sum of the reconnaissance times for each way point in addition to the traveling time in between each of the waypoints. This time to travel between each of the waypoints may or may not be negligible, depending on the specific context of the routing mission.

Meeting the time windows constraint for each way point in each route is a standard constraint of any VRP subject to the time windows constraint. Various techniques have addressed this constraint, but given a GA as the routing algorithm, two basic approaches are possible: to explicitly embed the time window information into the genotype or to use an auxiliary data structure to process this information. The standard fitness criteria for problems of VRPTW in the Solomon Test Set is the shortest duration of time required to complete all deliveries; the basic difference between the VRPTW fitness criteria and the CVRP fitness criteria is that the VRPTW fitness criteria must account for delays and delivery durations at each delivery location. There exist two ways in which the time window constraint may be violated: early arrival at a delivery location, or late arrival at a delivery location or the depot.

The best published results in EC literature use GVR [64]. Tavares handles the early arrival violation by simply having the vehicle wait until the earliest possible delivery time. The late arrival violation is handled by adding a new vehicle (and resultantly a new route) to the itinerary such that a new vehicle meets the delivery time window for the location that would otherwise have been in violation.

The threat constraints are fairly simple to incorporate into the standard VRP if a threat is viewed as an additive cost to the objective function for a way point. Chapter III discusses specific approaches that may be used for incorporating threats into a VRP with minimal impact on the algorithm's complexity. Handling changes in the threat environment at the subswarm level is simply a matter of rerouting the swarm amongst the remaining way points each time a significant change occurs.

4.3.2 Guaranteeing Feasibility with the Evolutionary Routing Algorithm. There exist three mainstream approaches for handling feasibility criteria with a GA. One approach is to generate an initial population of feasible individuals, and design operators that guarantee feasibility after their operation. The overwhelming disadvantage for this approach is that it is not always a straightforward process and can be difficult or impossible to implement efficiently for the crossover operator, especially with integer-represented solutions like in the TSP or VRP. Given this difficulty, repair and penalty functions become the notable alternative.

4.3.2.1 Repair Functions. A repair operator transforms an infeasible solution into a feasible one. This approach has the advantage over the previous possibility in that the crossover operator can be more simply and efficiently implemented. As a result, more creativity and thought can be placed on its design, and it can be more easily maintained or updated. A decision that must be made when designing a repair operator, however, is whether to repair explicitly as in a Lamarckian model, or implicitly as in a Baldwinian model. Still, there exists an uncountably infinite number of shades of grey in between, because a repair may occur with some probability $1 \leq p \leq 0$. Determining feasibility of a solution and repairing by segmentation can be accomplished with a simultaneous pass through a solution, and is of order $O(n)$.

4.3.2.2 Penalty Functions. A final possibility is not to pose any explicit feasibility constraints on a solution; rather, use a penalty function that guides the search toward feasible solutions by rating infeasible solutions with lower fitness values[1]. A penalty function should produce solutions to the problem that are feasible, and has the potential benefit over the repair function in that it does not explicitly manipulate the building blocks to guide the search. Still, effective penalty functions are usually not trivial to derive and often require problem specific information. Two common types of penalty functions include [6]

- Static Penalty Functions
- Dynamic Penalty Functions

[1]A broader categorization of penalty functions is to divide them into interior and exterior categories. Exterior penalty functions, however, are the topic of this discussion. Exterior penalty functions penalize infeasible solutions in an attempt to guide the search to the feasible solution space, while interior penalty functions penalize feasible solutions in efforts of finding a 'tight' solution–one that exists on the edge between feasibility and infeasibility [6]

Static penalty functions apply a constant penalty to solutions that violate feasibility in any way. One formulation of a static penalty function is given by Equation 4.1 [6].

$$f_p(x) = f(x) + \Sigma_{i=1}^{m} C_i \delta_i \qquad \begin{cases} \delta_i = 1 & \text{if constraint } i \text{ is violated} \\ \delta i = 0 & \text{otherwise} \end{cases} \tag{4.1}$$

A disadvantage of this approach, however, is that it assumes that the penalty is a function of the number of constraints violated, which may not be the case. For example, some optimization problem have few constraints that are difficult to satisfy. Additionally, suitable coefficients must be specified for this approach to be effective depending upon the relative severity of the constraints to one another. An additional concept to investigate when using a static penalty function like Equation 4.1 is whether or not penalizing for constraint violations can actually guide the search toward feasible areas.

Another approach that is generally superior to Equation 4.1 is one that penalizes according to the distance to feasibility or 'cost-to-completion' [31,54]. Equation 4.2 illustrates such a penalty function [6].

$$f_p(x) = f(x) + \Sigma_{i=1}^{m} C_i d_i^{\kappa} \qquad \begin{cases} d_i = \delta_i g_i(x) & \text{for } i = 1, ..., q \\ d_i = |h_i(x)| & \text{for } i = q+1, ..., m \end{cases} \tag{4.2}$$

In Equation 4.2, κ is a user-defined exponent, and d_i is the distance metric of constraint i applied to solution x. Constraints $1...q$ are inequality constraints and will be activated when the constraint is violated. Constraints $(q+1)...m$ are equality constraints that will activate the penalty if there is any distance between the solution and constraint values [6]. Thus, the inequality constraints guide the search toward the feasible solutions, and the equality constraints guide the search toward local or global optima. Although existing literature for static penalty functions is quite mature, defining parameters for penalty functions remains far from trivial and very much problem dependent. Additionally, the notion of 'distance' between feasible and infeasible solutions is not exactly straight-forward for hard optimization problems.

Adaptive penalty functions can be viewed as a particular implementation of Equation 4.2. Adaptive penalty functions increase penalty values in monotonically nondecreasing manner proportional to the search time, which may be the number of generations. Equation 4.3 illustrates an implementation of Equation 4.2, where $s_i(t)$ is the monotonically nondecreasing function that satisfies the definition of an adaptive penalty function as presented. An explicit example of $s_i(t)$ might be an expression such as $s_i(t) = C_i t$ or some variation thereof.

$$f_p(x,t) = f(x) + \Sigma_{i=1}^{m} s_i(t) d_i^{\kappa} \tag{4.3}$$

4.4 Summary

This chapter provided a mathematical model for a low level UAV routing algorithm using a Genetic Algorithm (GA) and the associated time complexities for its operators. The GA-driven routing algorithm may employ either a penalty or repair function to guide the search from infeasible to feasible search areas and can use a variety of mutators to explore the space. The GA's data representation allows for its crossover operator to better preserve good building blocks during the search than Path Based Representation approaches because it explicitly encodes each route separate from the others. The next chapter presents experiments for measuring efficiency and effectiveness metrics using this GA-driven routing algorithm.

V. Experimental Swarm VRP Design

"In whatever position you find yourself
determine first your objective."
– Marechal Ferdinand Foch

THE routing algorithm for a swarm simulation must be both efficient and effective. A typical method to measure effectiveness is running the algorithm on well-known benchmark problems and comparing the algorithm's results with the best published for the problems. Efficiency is often measured by the total search time, or time per iteration for an iterative algorithm. The most common benchmark problems for the VRP are approximately twenty years old and have been approached by nearly any algorithmic technique imaginable. For most of the benchmark problems, there are known optimal solutions and established efficiency metrics. This chapter outlines experiments for measuring the efficiency and effectiveness of a UAV Swarm routing algorithm. A swarm routing algorithm is the cornerstone of any mission planning system.

5.1 Experiments in Measuring GVR's Effectivness

This class of experiments is designed to provide insight into the effectiveness of GVR. These experiments collectively form a suite and the outcome of each experiment successively determines a point of interest for additional experiments. This is to say that preliminary experiments are somewhat broadly focused, and each successive experiment gains slightly more focus based on previous results from experiments in the suite.

5.2 Experiments in Routing

Experiments in this suite are designed to provide insight into GVR's efficiency and effectiveness. These experiments collectively form a suite and the outcome of each experiment successively determines a point of interest for the following experiment. This is to say that preliminary experiments are somewhat broadly designed, and each successive experiment becomes more narrowly focused based on previous results. For implementation, the GA is designed with GAlib [71] and run on an AMD Opteron Processor running at 2.2GHz and 4GB of memory. A beowulf cluster is used to facilitate and speed up the data collection process. Enough runs and data are collected to either qualitatively or statistically fulfill each experiment's objective.

Experiment 1.

Objective 5.1. *To qualitatively determine the extent that a static penalty function can guide the search process and lead to effective solutions using GVR.*

The objective function is crucial to successful search with a GA. An important design decision impacting a GA's effectiveness is how it handles infeasible solutions generated from the recombination operators. GVR very often generates infeasible solutions from the crossover operator; the infeasible solutions could be either repaired or penalized in order to guide the search. Previous approaches for GVR used repair functions for guiding the search [51, 62–64].The intention of this experiment is to determine to what extent a static penalty function can guide the search process. Feasible CVRP solutions pose two basic constraints: 1) No vehicle capacities can be exceeded, and 2) At least the minimum number of routes possible must appear in a solution.

The problem definition provides vehicle capacities, and a lower limit on the number of possible routes is given by dividing the total demand of all delivery locations by the vehicle capacity. Equation 5.1 penalizes both criteria; for the former condition, it penalizes using a 'cost-to-completion' metric, which is shown to be more effective than penalizing based only on the number of violated constraints [6].

$$f_p = \Sigma_{i=1}^{R} f_{cap} \cdot f_{veh} \qquad \begin{cases} f_{cap} = P_c l_i \delta_c \\ f_{veh} = P_v \delta_v \\ \{\delta_c, \delta_v = 0\} \quad \text{if i satisfied. } \{\delta_c, \delta_v = 1\} \text{ otherwise} \end{cases} \tag{5.1}$$

This experiment runs all possible ratios of penalties, p_{cap} and $p_{veh} \in \{0, 1, 2, 4, 8, 16, 32, 64, 128, 256\}$ on benchmark problem A-n32-k5 [8]. This problem, as previously mentioned, is of relatively low combinatorics and is optimally solved in well under 10,000 generations by previous GVR research using a repair-based approach [62,63].

Experiment 2.

Objective 5.2. *To measure the effectiveness of the repair operator in guiding the search process.*

The GA is run on a diverse test suite of common CVRP benchmarks using a repair operator that segments infeasible solutions at the point of violation in the route. [1] The benchmark problems include A-n32-k5, A-n54-k7, A-n69-k9, B-n63-k10, E-n76-k8, and M-n200-k17. Results are compared to the best published in the literature and to results from other algorithmic techniques.

For all problems except A-n32-k5 and M-n200-k17, population sizes of 200 with 50,000 generations of evolution are tested. For problem A-n32-k5 only 10,000 generations are tested, because this is an 'easy' benchmark problem and most published results use this metric. For problem M-n200-k17, a population of size 400 and 100,000 generations are used because the combinatorics of the problem are much larger than the other benchmarks under test. Furthermore, a matrix of runs is conducted for all possible combinations of $p_{repair} \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$ and for all possible mutation combinations from $p_{mutation} \in \{0.05, 0.10, 0.15, 0.20\}$ except for two cases. Because problem M-n200-k17 is very large and takes a long time period to execute, the mutation combinations are reduced to $p_{mutation} \in \{0.10, 0.15, 0.20\}$. For this experiment, tournament selection with tournament size $q = 5$ and crossover $p_{cross} = 0.75$ is used. The tournament size is designed to apply very little selection pressure during the search, while the crossover rate is intended to exploit the effects of recombination. Given various solutions to these benchmark problems, visualization is accomplished via graphing the maximum, average, and minimum fitness values over time.

[1]Experiment 4 compares this operator to one that attempts to pack routes as tightly as possible by fitting any possible locations into the remaining space.

Experiment 3.

Objective 5.3. *To determine how dependent effective problem solutions depend on tuning mutation parameters.*

Effective mutators are crucial to gaining insight into a GA's ability to explore. Given the best results from problem M-n200-k17 in Experiment 2, four matrices of runs with fixed crossover rate and variable mutation rates are constructed. Graphical display of the data is accomplished by holding two of the mutators at constant values while varying the others over a fixed range for 0.02 increments. This grid size should be fine grained enough to collect data indicative of the landscape. The combinations of rates being varied include: displacement and insertion, swap and inversion, inversion and displacement, and swap and displacement.

Experiment 4.

Objective 5.4. *To determine if there exists a statistical significance between two different repair operators; one operator segments routes at the point of capacity violation, while the other packs the existing route as tightly as possible.*

Given the best results from problem M-n200-k17 in Experiment 2, this experiment determines whether there is a significant difference in solution quality for two different repair operators. One operator repairs by packing the infeasible routes encountered during objective function evaluation as tightly as possible. Another repair operator, the one used in Experiment 2 simply segments infeasible solutions at the point of the violation and creates another route with the infeasible portion. For each of the two operators, a series of 11 runs with different random number seeds is conducted.

Experiment 5.

Objective 5.5. *To determine how dependent the search process is on tournament size and if population reinitialization after a time of stagnation can improve search results.*

Selection pressure is proportional to the ratio of the tournament size q to the population size. Given graphical displays of maximum, mean, and minimum fitness values over time from Experiment 2, this experiment determines if doubling q and reinitializing the population each time the minimum fitness value in the population does not decrease by more than $\epsilon = 10$ units can improve solution quality. This approach is tried on the best and worst solutions for problem M-n200-k17 from Experiment 2.

Experiment 6.

Objective 5.6. *To determine if high quality solutions generated from the GA can be further improved by a lambda exchange local search algorithm [65] as a post-processor.*

Given a high quality solution produced with the Genetic Algorithm, this experiment determines if the results can be further improved by means of a fast local search that systematically displaces all combinations of up to two locations. Results from this experiment provide additional insight into the proximity of 'good' solutions to either local or global minima.

5.3 Experiments in Simulation

Experiments 7-10 compute and analyze the performance of a parallel swarm simulation employing Kadrovach's swarm model [39], that is parallelized by Corner [16]. The primary objective of these experiments is to measure and improve the parallel simulation's performance.

Experiment 7.

Objective 5.7. *To measure runtimes and calculate speedup and efficiency metrics for small swarms,* $|SWARM| \in 32, 64, 128, 256, 512$*, on varying numbers of processors,* $P \in 1, 2, 4, 8, 16, 32$ *using* optimistic *event processing.*

The underlying motive for this experiment is to be able to reproduce data from Corner's research [16] validating the lack of speedup for even small swarm sizes on various CPU configurations. The results of this experiment allow additional experiments in the suite to occur that make comparisons between optimistic and conservative time management algorithms.

Given the standard parallel computing metrics for small swarm sizes, some insight is gained as to how larger swarms may perform in parallel simulation under the same optimistic time management algorithm. Corner's previous parallel swarm research showed no speedup for any configuration with any swarm size using the SPEEDES libraries; reproducing these results is the first step toward future AFIT parallel swarm research.

Experiment 8.

Objective 5.8. *To compare runtimes between conservative and optimistic techniques using the SPEEDES libraries for various swarm sizes,* $|SWARM| \in 20, 40, 60, ..., 160, 180, 200$*, on a single processor.*

Previous research by Corner [16] implicity hypothesized that the Breathing Time Warp Algorithm results in a more efficient simulation[2], but his research results did not confirm this hypothesis. In fact, his experimental results showed that for small simulations involving only 500 UAVs and 12 simulation time slices, there were more than 1.6 million rollbacks and the ratio of simulation time units to wall clock time varied between ratios of 1:1000 and 1:2000.

This experiment is designed to verify that there are no significant efficiency differences between conservative and optimistic techniques for runs on a single processor. Theoretically, there are not expected to be any significant differences because no rollbacks should occur for the optimistic technique on a single processor, thereby having it behave the same as a conservative one. If there exist any significant overheads of using an optimistic algorithm in a simulation using SPEEDES, results from this experiment should indicate that overhead.

[2]In theory, optimistic techniques often do provide superior results to conservative ones, and the SPEEDES User's Guide makes a good case for using the Breathing Time Warp Algorithm.

Experiment 9.

Objective 5.9. *To compare runtimes for conservative processing techniques using varying lookahead window values* $W \in 0, 3, 4, 5, 6, 7$ *with the SPEEDES libraries for various swarm sizes,* $|SWARM| \in 50, 55, 60, ..., 95, 100$*, on 5 processors.*

This experiment follows the same procesure as Experiment 8, except that the runs are accomplished on multiple processors and for various lookahead windows. Results from this experiment should provide insight into the sensitivity of the lookahead value, and the gain realized by using a lookahead window as opposed to not using one.

Experiment 10.

Objective 5.10. *To compare the difference in conservative and optimistic time management schemes for a parallel UAV swarming application using variable CPUs and swarm sizes.*

This experiment calculates the standard speedup and efficiency metrics for a conservative time management event processing scheme with the SPEEDES libraries and compares them with results from the optimistic event processing results from Experiment 7.

Given a particular lookahead window, the standard parallel performance metrics are calculated and compared to results from optimistic event processing using the SPEEDES Breathing Time Warp algorithm. Recommendations are made about future direction in future parallel swarm simulation based on the relative differences in results from the two techniques.

5.3.1 Summary of Experimentation. Experiments in this research endeavor primarily involve two endeavors: 1) providing an efficient and effective routing algorithm that can be synthesized into existing AFIT UAV research, and 2) improving parallel performance for the AFIT UAV swarm simulation. A high level discussion reviewing experimental results from this entire suite is crucial to synthesizing the results into a single unified message relevant to the overall research objectives as discussed in Chapter 1 and to AFIT swarm research.

5.4 Facilitating Experimentation with Software Tools

5.4.1 GAlib.

Implementing experiments can be quite cumbersome without adequate software tools. Often times, existing software applications are modified or tailored for a new purpose, suites are developed from scratch, or ad-hoc components are melded together in a fragile manner. None of these scenarios is ideal. A much more desirable alternative is to use a well-tested and well-documented library of components for implementation. The use of such an existing library reduces development time, provides maximum flexibility for change and alterations, and allows developers to focus on the core of the experiment instead of the implementation details. GAlib provides this opportunity.

GAlib is a C++ library of Genetic Algorithm components written by Matthew Wall from the Massachusetts Institute of Technology. The software is free open source software, but subject to some distribution and not-for-profit constraints [71]. The suite is a well-tested and widely used library that provides the basic framework for developing an executing a GA. Basic features of the library include predefined chromosome representations, selection mechanisms, crossover operators, mutators, automated statistics collection, and support for Parallel Virtual Machine (PVM).

In addition to its built in features, GAlib is very customizable. Individual components can be modified, new chromosome representations can be defined, and completely tailored GAs can be produced in far less time than developing from scratch. To develop and execute a GA for these experiments, the following steps are necessary:

- Download and compile the software
- Define a new chromosome representation for GVR
- Define recombination, population initialization, and utility operators for GVR
- Tailor any existing components such as selection mechanisms, etc.
- Develop any auxiliary components such as benchmark file readers, etc.

Defining the custom chromosome representation is by far the most time consuming step. Given successful completion of the previous steps, built-in GAlib components provide a rich set of options for experimentation such as:

- The type of GA: Simple GA, Steady State GA, Island Model GA, etc.
- The type of selection operator: Tournament, Proportional, Random, etc.
- The collection of statistics and diversity data
- Convergence criteria

For a complete list of GAlib's features see the User's Guide [71]. Auxiliary code, compilation notes, and a build script is given in Appendix I

5.5 Summary

This chapter presented experiments designed to measure efficiency and effectiveness of a specific routing algorithm implementation that employs a Genetic Algorithm. Experiments are tailored to reveal characteristics of the search that apply specifically to an "on demand" mission planning system that expects high-quality results as quickly as possible and for which the amount of time to search the space is variable. The next chapter presents the results of this experimentation suite and analysis.

VI. Experimental Results and Analysis of Swarm VRP

In questions of science the authority of a thousand is
not worth the humble reasoning of a single individual.
–Galileo

EXPERIMENTATION without analysis is a futile activity. Demonstrating the existence or lack of effectiveness and efficiency for an algorithm requires transforming the experimental data into meaningful information and performing qualitative analysis.

6.1 Approximation of VRP and GVR Combinatorics

In order to appreciate the importance of effective search, a brief discussion and analysis of VRP combinatorics is presented.

6.1.1 Mathematical Approximation. Although knowing that the VRP is at least as hard as the TSP does provide some insight, it is useful to annotate a more precise mathematical analysis of the problem's combinatorics and how the algorithm maps data structures to individual solutions in the overall solution space.

Illustrating that the VRP is at least as hard as the TSP is trivial. Consider the case where there exists only 1 vehicle to service $|L|$ delivery locations; the optimal solution is one of $|L|!$ possible Hamiltonian circuits.

One possible approach to illustrate the mapping of VRP solutions to the solution space GVR represents is to consider all possible partitionings for all of the possible Hamiltonian circuits given the $|L|$ delivery locations. For any given Hamiltonian circuit of $|L|$ locations there are exactly $\binom{|L|}{|L|}$ ways to leave the circuit as is, $\binom{|L|}{|L|-1}$ possible ways to segment the circuit into two partitions, $\binom{|L|}{|L|-2}$ ways for three partitions, and so forth. Equation 6.1 gives the general case for determining all possible partitionings of $|L|$

$$P = \Sigma_{i=1}^{|L|} \binom{|L|}{i} = 2^{|L|} - 1 \tag{6.1}$$

Given that proving an optimal solution for the VRP requires consideration of all possible partitions for all possible permutations of $|L|$, the VRP is harder than TSP by at least a factor of $2^n - 1$, given by Equation 6.1. Conceptually, for every one of the $|L|!$ possible Hamiltonian circuits, P possible partitionings must be considered; to phrase it differently, all possible subsets of $|L|$ must be considered. The GVR solution space is capable of representing duplicate solutions to include the solutions that differ only because the routes are ordered differently. Certainly not all subsets of the power set of solutions is in the feasible solution space for the VRP, but they still represent the entirety of the problem's combinatorics and comprise the space GVR can represent.

6.1.2 The Importance of Effective Search. To gain perspective on the combinatorics, consider a pedagogical example with a TSP having $|L| = 11$. There exist $3,628,800$ possible Hamiltonian circuits, at least one of which is an optimal solution. Now consider the VRP with $|L|$, where the depot is included in L. Substituting $|L| = 10$ into Equation 6.1 yields $P = 1023$. Thus for each of the $3,628,800$ Hamiltonian circuits, an additional 1023 partitionings must be considered for a total of $3,715,891,199$ possible solutions. Although all of the $|L|!$ Hamiltonian circuits are distinct, this approximation includes duplicates which often appear in the search space for GAs like GVR and other stochastic searches.

For a simple benchmark problem, A-n32-k5, $|L|! = 2.63130836 \times 10^{35}$ and $P = 4,294,967,295$. Equation 6.2 is a rough approximation for the combinatorics of the problem. For large benchmark problems, the combinatorics quickly become intractable. Even if evaluating each possible solution only requires 1 cycle on a 1GHz processor, $8.34382410368 \times 10^{21}$ years are required to evaluate all possible solutions in order to *prove* solution optimality. Parallel computing metrics can only help

Table 6.1: Summary of benchmark results from Experiment 2

Problem	p_{rep}	p_{swp}	p_{inv}	p_{dis}	p_{ins}	fitn.	best	% diff
A-n32-k5	**	**	**	**	**	784	784	0
A-n54-k7	**	**	**	**	**	1167	1167	0
A-n69-k9	0.5	0.15	0.15	0.15	0.1	1164	1159	0.4
B-n63-k10	0.5	0.05	0.15	0.1	0.1	1507	1496	0.7
E-n76-k8	0	0.15	0.1	0.1	0.05	741	735	0.8
M-n200-k17	0.5	0.1	0.15	0.1	0.1	1348	1296	3.9

by a constant factor p, decreasing that amount of time proportionally to the number of processors.

$$Com_{VRP} = |L|! \cdot 2^{|L|} - 1 \tag{6.2}$$

6.2 Results from Experiments in Routing

Given the importance of effective search techniques and the combinatorics of the VRP, experiments measuring GVR's effectiveness are now presented.

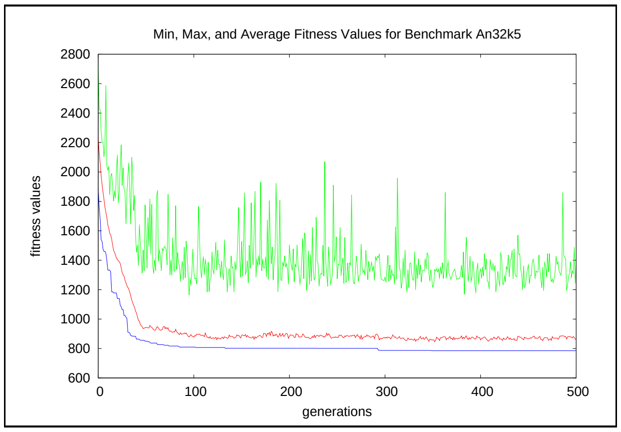
Experiment 1. For all ratios of penalty functions given by Equation 5.1, no solutions were feasible. Given that problem A-n32-k5 is considered an 'easy' benchmark problem, these results suggest at a minimum that Equation 5.1 is not an effective static penalty function, and more likely, that static penalty functions are not effective techniques for guiding the search process. Results from Experiment 2 produce results competitive with the best-published results on A-n32-k5 in under 300 generations. It appears that penalizing solutions to the problem is not effective in part because of the sheer combinatorics. All other experiments in this suite use repair functions to guide the search process. Future work in investigating penalty functions should inspect adaptive or dynamic approaches [6].

Experiment 2. The GA produces optimal or near optimal results for the CVRP benchmarks tested. All results are more than adequate for effective vehicle routing with very little time for *a priori* computation. Known optimums were quickly reached for all tested parameter combinations in problems A-n32-k5 and A-n54-k7. Near optimal and high quality solutions for all other parameter combinations were produced for all other problems.

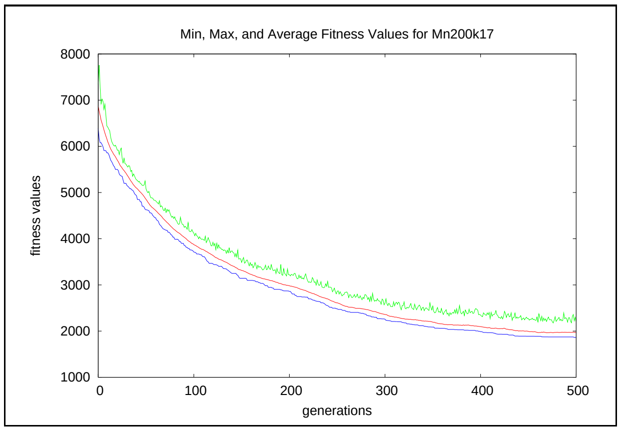
The high quality results GVR produces are desirable, but it is important to determine how fast the search converges on these results if it is to provide the engine for a route planning system that demands near optimal solutions as quickly as possible. Mission planning does not necessarily entail an optimum solution; rather, solutions within a particular tolerance are often acceptable. Additionally, it might be noted that the exponential increase in solution quality suggests that this algorithmic technique is likely to provide competitive results for instances of the Dynamic VRP, in which delivery schedules might change while vehicles are en route. A graphical display of the search progress over time reveals that a period of rapid improvement in solution quality is followed by long periods of stagnation for all benchmark problems. In the end, this type of convergence is almost ideal for fast *a priori* routing. This suggests that a local search or a similar technique could be used during this period of stagnation in lieu of continuing the global search with the GA, although the GA does continue to gradually improve results. It is also evident that GVR increases the number of good building blocks at an exponential rate, per the Schema Theorem[1] [72]. Gradually improving partial solutions in a problem like the VRP is more than reasonable; locations close together gradually link into routes, and routes that complement one another have a higher probability of survival and continued improvement. The results from Experiment 5 presents the results from increasing the selection pressure and reinitializing the population during these periods. Results from Experiment 6 present the results from trying a local search as a post-processing routine.

Figure 6.1 shows the max, min, and average fitness values of each generation for benchmark problems A-n32-k5 and Mn-200-k17. Problems A-n63-k10, A-n69-k9, B-n63-k10, and E-n76-k8 result in the same net effect: a period of rapid increase in solution quality is followed by a period of stagnation with very little improvements occurring. Only the first 500 generations are shown.

[1]Acceptance of the Schema Theorem is certainly not unanimous [67].



(a) Generations 1-500 of A-n32-k5



(b) Generations 1-500 of M-n200-k17

Figure 6.1: Solution quality exponentially increases

Note that the elitism employed in the GA results in a monotonically non-increasing minimum fitness value over time. The fluctuations in the maximum and average fitness values illustrates that exploration is taking place, but the stagnation of minimum fitness values indicates that the search space is treacherous once solutions become nearly optimal. A fair description of the search space at this low level amounts to something like trying to sink a golf ball into a hole surrounded by potholes and ruts from 100 yards away using only a bent putter.

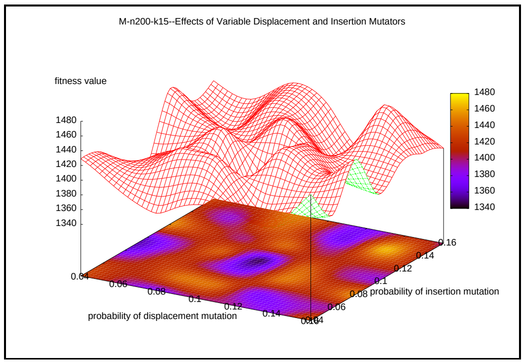
Table 6.2: Search times for selected benchmarks based upon the search time per generation and a subjective search termination based upon stagnation trends from the fitness over time graphs.

Benchmark	sec/gen	termination after	routing time
A-n32-k5	0.001423	500	7.15 sec
A-n69-k9	0.02916	5000	2.43 min
B-n63-k10	0.0277	10000	4.61 min
E-n76-k8	0.0311	10000	5.18 min
M-n200-k17	0.158	15000	39.5 min

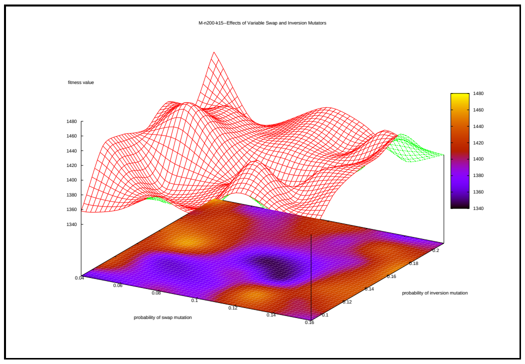
Table 6.2 breaks down the computation time required for each generation and hypothesizes an *a priori* routing time based upon subjectively terminating the search after the search begins to stagnate. The calculated times per generation are calculated by dividing an entire search time by the number of generations and taking the average. The runs were executed on a 32-bit AMD Athlon processor running at 2.2 GHz with 512kB of cache and 1GB of memory.

Given a 'man-in-the-loop' controller or even a primitive threshold condition to detect stagnation, the GA produces exceptional routing times and high quality solutions for very difficult NP-complete CVRP benchmark problems.

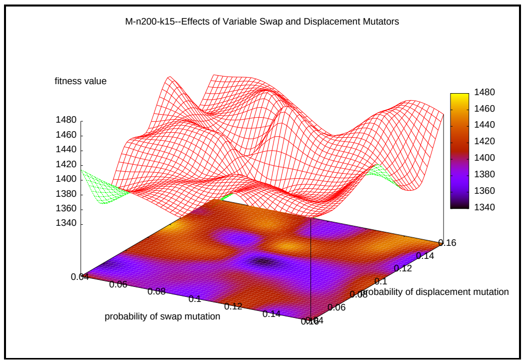
Experiment 3. This experiment varied two mutation rates over a fixed range in incremental amounts while holding the other two mutation rates constant for the best results on problem M-n200-k17 in Experiment 2. The results of the experiment as shown in Figure 6.2 simply provide a visual display of the problem landscape as a projection of two dimensions. Portions of the surface near the front of the plot are not displayed if it were to block any part of the entire contour map, and consequently, cause data loss in the graph. Very small adjustments to mutation rates can impact the quality of solution discovered by GVR. This insight confirms the idea that the landscape searched is very jagged and that local minimal run rampant. Global minima are very difficult to find, even though the local minima are not very different in terms of percentage difference. For mission routing purposes, sensitivities to mutation parameters might be overcome by having several processors compute routes using different mutation parameters in parallel, or the mutators might be encoded as part of the chromosome in an effort to become self-tuning.



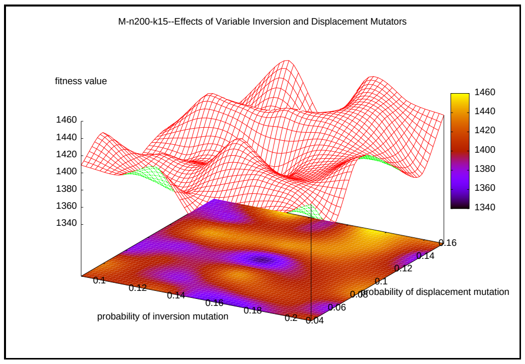
(a) Displacement and Insertion



(b) Swap and Inversion



(c) Swap and Displacement



(d) Inversion and Displacement

Figure 6.2: Effects of varying two mutation rates while holding the others constant.

Experiment 4. Table 6.3 illustrates the results of running 11 different seeded runs with the best results from problem M-n200-k17. Segmentation packing produces better results in 6 of the 11 trials, but a Student's T-test with $\alpha = 0.05$ reveals that there is not a statistical significance between the two packing methods. Intuitively, it seems that the segmentation packing method should have produced superior results, because it does not disrupt the good building blocks. A possible explanation that the two methods are not significantly different, however, might be that the recombination operators sufficiently exploit the solution space enough to compensate for any disrupted building blocks.

Table 6.3: Results from vehicle packings

Seed	Tight	Segmentation
1900	1385	1430
2003	1441	1445
2020	1424	**1348**
3131	1408	1387
4242	1387	1366
5353	1428	1360
5555	1420	1390
6038	1443	1387
7777	1444	1445
8765	1385	1394
9843	1389	1391
AVERAGE	1414	1394.818182
STD DEV	24.2363364	32.66440932

Table 6.4: Reinitializing does not improve results.

	Seed 2020		Seed 3131		Seed 4242	
	4000	{8000,12000}	4000	{8000,12000}	4000	{8000,12000}
{5,10,15,20}	1359	1345	1412	1395	1361	1375

Experiment 5. Experiment 2 produced a solution of value 1348 for problem M-n200-k17 without reinitialization. Reinitializing with the same random number seed produced a solution of lower quality in all cases except for one, where the improvement was negligibly improved to 1345. This indicates that 'good' partial solutions, when continually exploited, can continue to improve to some degree by converging to local minima. Reinitialization disrupts the good building blocks, even though it provides the opportunity for more of the search space to be examined. This experiment, like the previous ones, provides insight into the jaggedness of the search space and the abundance of local minima. A viable routing algorithm demands a quick high-quality solution, which is most likely not an optimum solution. Other approaches to improve solution quality should be tried, but in the interim, existing GVR solutions are still more than adequate for a UAV routing application.

Experiment 6. In all cases, the local search algorithm employed did not improve solution quality. The most likely reason is because the fitness landscape, as previously discussed, is especially treacherous once solutions approach near optimality. There are concentric layers of myriad local minima scattered throughout the space that are high-quality even when compared to the optimum.

It may be difficult even for a lambda exchange algorithm or other local search technique to improve such near-optimal solutions.

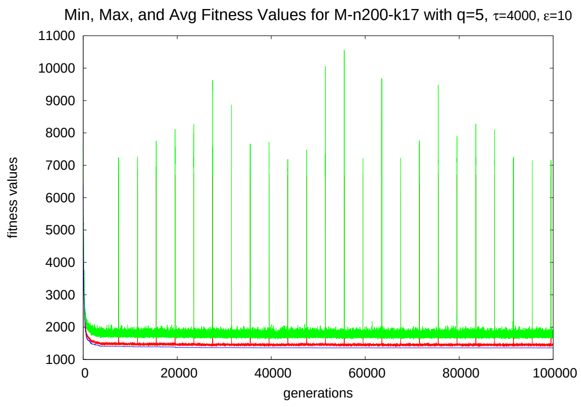
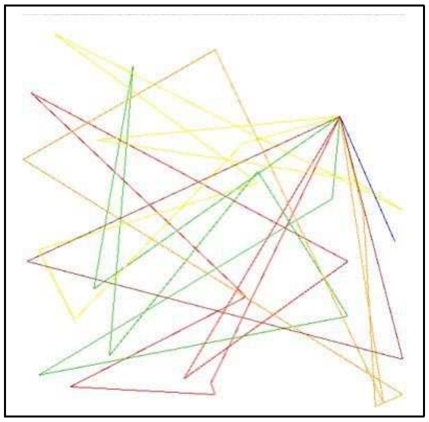


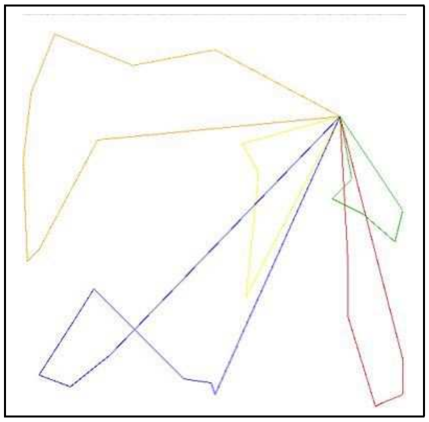
Figure 6.3: Reinitializations over time.

By the ‘No Free Lunch Theorem’ [73], there is no single search technique that can work for all problems in all cases. Consequently, the most important meta-level knowledge about search is to understand and adapt to the search landscape. A crude measure of the search landscape is known about the CVRP just by virtue of it being an NP-Complete problem. More specific insight about the search landscape is alluded to in Experiment 3 via the fairly high fluctuations in solution quality by changing the GA’s mutation parameters. Even more precise information about the search landscape can be gleaned by examining partial solutions to the problem in incremental time steps. Particular consideration should be given to partial solutions obtained during times of stagnation, and final solutions obtained from the search should be compared to proven optimal solutions if available.

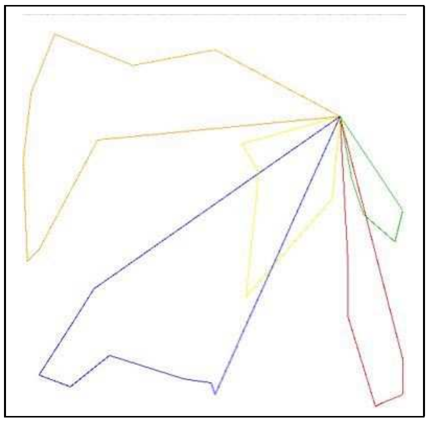
Figure 6.4 shows the landscape for benchmark A-n32-k5 at interesting points during the search. Note that for this problem of small size, the number of local minima is quite large and that most of the time searching is spent escaping from the concentric layers of local minima. Good partial solutions quickly emerge, but most of the search time is spent overcoming various layers of local minima.



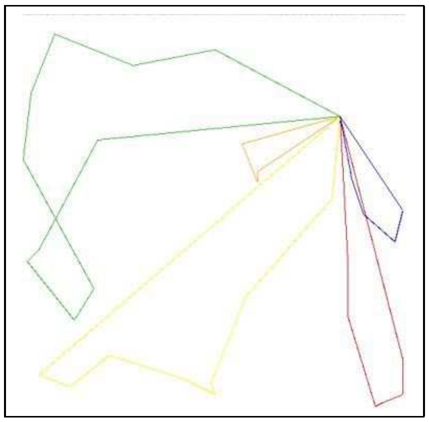
(a) Generation 1



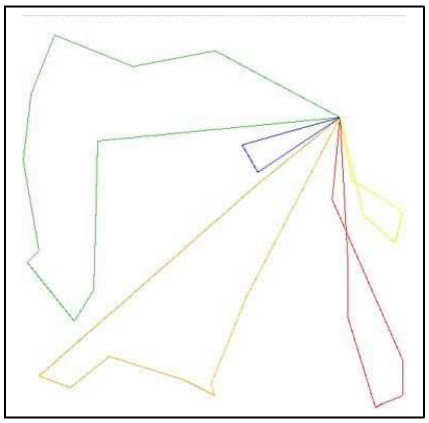
(b) Generation 121



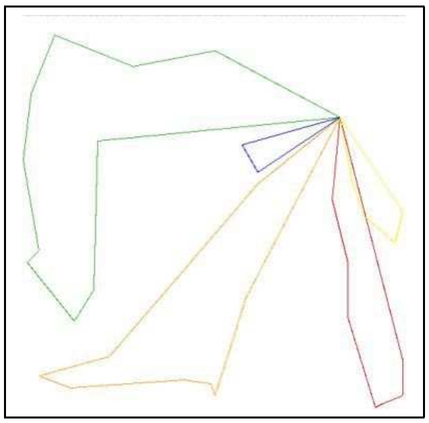
(c) Generation 229



(d) Generation 940



(e) Generation 1180



(f) Generation 2100

Figure 6.4: Snapshots of partial solutions for benchmark problem A-n32-k5.

6.3 Results from Experiments in Simulation

Experiment 7. This experiment used UAV swarm sizes $|SWARM| \in \{32, 64, 128, 256\}$ and CPU sizes that are powers of 2 so that the UAVs per CPU could divide evenly to measure performance of the BTB algorithm. Given previous results from Corner [16], optimistic processing of UAV sizes much larger than these swarm sizes is possible because of the sheer number of rollbacks and long run times that occur using optimistic time management via the Breathing Time Warp (BTW) or Breathing Time Buckets (BTB) algorithms from the SPEEDES libraries.

Experimentation in this suite confirmed that the simulation currently cannot produce results for swarm sizes of 512 or larger using optimistic event processing. This is very likely explained by the excessive memory required by SPEEDES for optimistic event processing, in part because of the overhead involved with rolling back events. Manually inspecting memory usage using standard system commands such as "top" for CPUs running ratios of more than 32 CPUs reveals that eventually the entire node's memory is used up by the "pswarm" and "ExternalModule" processes, and the node eventually crashes. Results can probably be obtained by running much smaller simulation durations, but it is unlikely that reliable steady state results can be obtained for much less than 12 simulation time units [16]. Even if results could be obtained for less than this steady state threshold, it would provide almost no benefit for simulation purposes; any realistic UAV simulation contains at least thousands of time increments.

Speedup and efficiency calculations for this experiment define the best serial runtimes as the ones from using SPEEDES with a single CPU. It is important to note that Corner's speedup calculations used serial run times from the original Linux port of 'cline.' Corner was not able to produce any speedup using the best serial runtimes as the ones shown in Table 6.5. This conundrum can begin to be explained by recalling that the inherent swarm model taken from Kadrovach's Ph.D research was designed using a global data structure in which all swarm members communicated with one another, yielding $O(|SWARM|^2)$ communications at each step. Given the

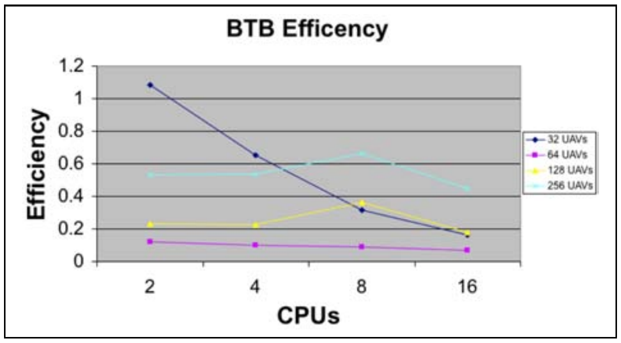
Table 6.5: Results from page 94 of Corner's research [16], citing the best serial runtimes for swarm simulation on the Linux port of 'cline.' Parallel runtimes using the SPEEDES library do not even begin to approach these runtimes even in the best cases.

UAV Count	Execution Time (sec)
100	4
500	198
1000	1478

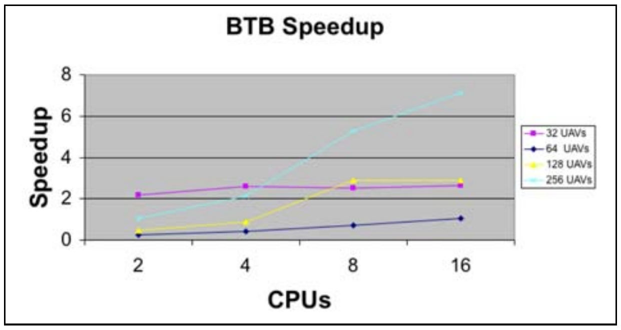
aforementioned algorithm designed as a serial application, *empirical results give no hope whatsoever that a parallelization of the serial model as SPEEDES libraries can ever produce any speedup.* Experiments 8-10 in the remainder of this test suite continue to calculate speedup and efficiency using the runtimes from the SPEEDES application on a single CPU as the base.

Results from this experiment indicate that for the given swarm configurations and for 12 simulation time units, the wall clock time can be decreased, speedup can be gained using BTB and an 'apples to apples' comparison. Figure 5(c) shows the runtimes from a matrix of runs on various processors. Figures 5(b) and 5(a) show the speedup and efficiency trends.

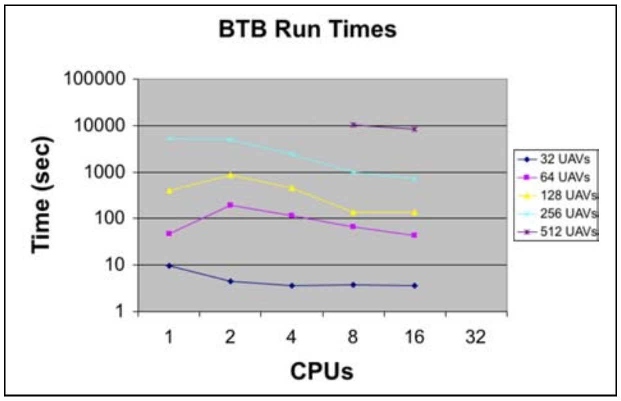
Somewhat unexpected is the increase in the run times when transitioning from 1 to 2 CPUs. This is likely explained by the communication inefficiencies introduced because of the proxies used within the SPEEDES libraries and is consistent with the trends found in Corner's research [16]. Transitioning from 2 to 4 CPUS probably relieves some of the administrative burden of overhead, thereby decreasing some of the communication inefficiencies introduced by the proxies and allowing some speedup to occur.



(a) Efficiency



(b) Speedup



(c) Run Times

Figure 6.5: Results show that for a small swarm of UAVs, there is eventually only negligible gain for parallelizing the swarm algorithm using the SPEEDES library. For a larger swarm size, however, there is significant gain, but efficiency quickly becomes a predominant concern.

Table 6.6: Comparison of wall clock times for simulating various swarm sizes on a single CPU using conservative and optimistic time management with SPEEDES.

—SWARM—	conservative	optimistic	% diff
20	3.77	5.23	27.91
40	13.35	15.67	14.81
60	39.17	40.58	3.47
80	86.73	88.84	2.37
100	170.79	176.37	3.16
120	312.03	326.49	4.42
140	534.98	551.74	3.03
160	860.05	886.45	2.98
180	1326.09	1382.55	4.08
200	1963.76	2053.95	4.39

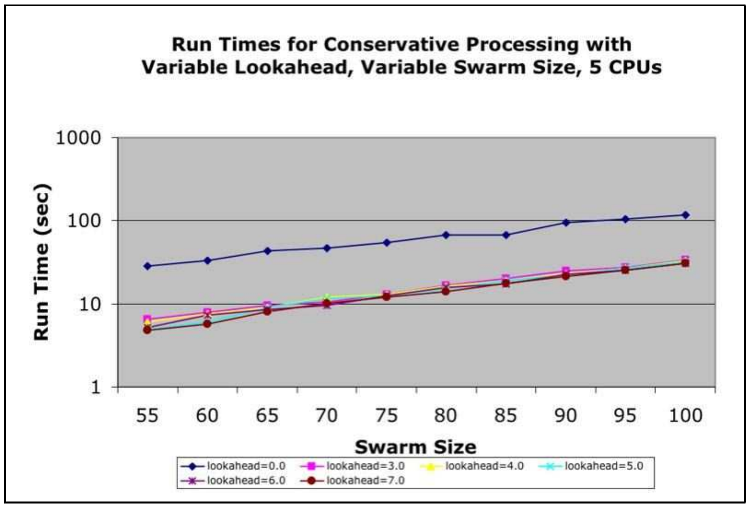
Experiment 8. This experiment measures the difference between using conservative and optimistic techniques for identical runs of a SPEEDES application on a single CPU. As expected, results show that there is some additional overhead for using an optimistic processing technique, but this overhead is small enough to be considered negligible. For the smallest swarm sizes $|SWARM| \in 20, 40$, the percent difference is quite large, but only because of a constant overhead imposed by starting up and configuring the SPEEDES server in comparison to the very short runtime for such a small swarm size. Results from this experiment verify that there is not a significant overhead imposed by SPEEDES for optimistic time management, and in particular, for the BTB. Given BTB's small overhead, it is likely that performance issues alluded to by Corner are *primarily* the results of excessive rollbacks. In Corner's own words from "Experiment P2" [16]:

> "Obvious eyesores are the millions of rollbacks that are occurring on a regular basis. This is probably due to the optimistic processing of future events that need rolled back because each UAV depends on the data that has already been modified at a future simulation time, so that n UAVs are causing rollbacks on all other UAVs.

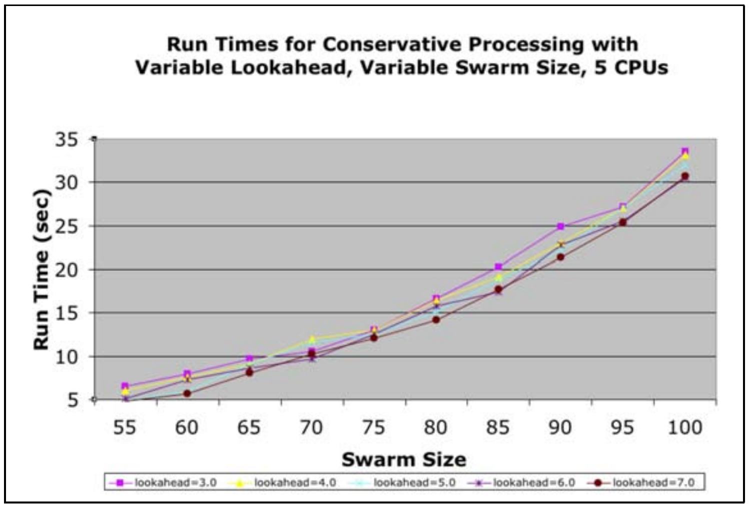
The final experiments in this suite are explicitly measure and compare performance between conservative and optimistic even processing with SPEEDES.

Experiment 9. This experiment compares the results for different lookahead window values for a conservative time scheme with SPEEDES to determine the general sensitivity to the lookahead value, as well as to determining the impact of not having a lookahead value at all.

Overall results are given in Figure 6.6 and indicate using even small values for a lookahead window significantly decrease the runtime for conservative event processing in comparison to using a lookahead value of 0.0. Difference in lookahead values can be explicitly compared using the values from Table 6.7.



(a) All Conservative Runs on a logarithmic scale



(b) The tightly grouped series of runs on a standard linear scale.

Figure 6.6: Results from a series of runs using 5 CPUs with various lookahead window values.

Table 6.7: Explicit runtimes from running various swarm sizes on 5 CPUs with various lookahead values.

—SWARM—	lookahead 0.0	lookahead 3.0	lookahead 4.0	lookahead 5.0	lookahead 6.0	lookahead 7.0
55	28.26	6.57	6.05	4.92	5.14	4.78
60	33.21	7.97	7.54	5.99	7.28	5.66
65	43.66	9.65	9.08	9.11	8.6	8.03
70	46.59	10.5	12.01	11.59	9.65	10.22
75	54.13	12.99	12.98	12.57	12.54	12.07
80	67.97	16.68	16.48	15.14	15.83	14.15
85	67.98	20.29	19.16	18.89	17.39	17.73
90	94.59	24.83	23	22.17	22.8	21.29
95	103.69	27.21	27.06	27.05	25.55	25.38
100	117.25	33.57	33.19	31.92	30.52	30.66

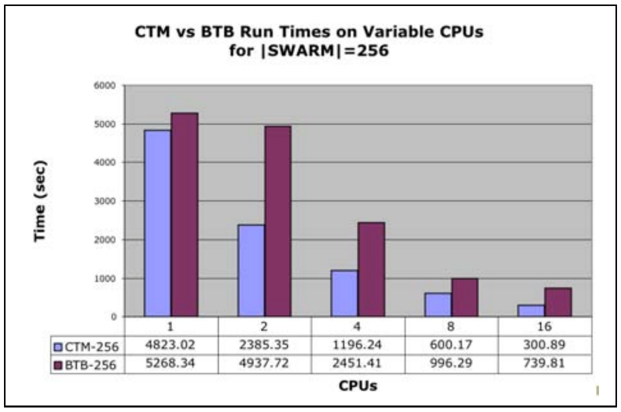
Experiment 10. Given that an optimistic time management technique does not impose a significant overhead when compared to a conservative technique, and that results from optimistic time management for a UAV swarming application with SPEEDES are less than impressive, an objective comparison is made between conservative and optimistic SPEEDES time management techniques.

The runtimes, speedup, and efficiency graphs from a series of conservative runs comparable to the optimistic runs using BTB from Experiment 7 are given in 6.8.

The metrics from the conservative runs and shown in this experiment exemplify the 'regular' behavior expected by adding additional processors. In particular, the runtimes decrease in all cases by the addition of more CPUs as show in Figure 8(c); there is observable speedup from the additional CPUs, and for these particular runs, the number of additional CPUs did not become great enough to begin leveling out the speedup curve in Figure 8(b); and finally, the efficiency decreased within an acceptable tolerance that could have been expected.

In fact, the only abnormalities with the data collected using CTM are 'good' abnormalities– a slight superlinear speedup that can be observed for the largest swarm size of 512 UAVs. The superlinear speedup in this case is most likely explained by hardware features that put the serial implementation at a disadvantage [33]. In this case, the limiting hardware feature is probably the memory limitations from processing 512 UAVs on a single CPU. Not given enough memory, the CPU would have had to write and read data from disk periodically, an operation that can take hundreds of times the duration from accessing memory in the worst case. Dividing the swarm size between 16 CPUs, however, would have eliminated this ‘problem’, thus resulting an perceived superlinear speedup.

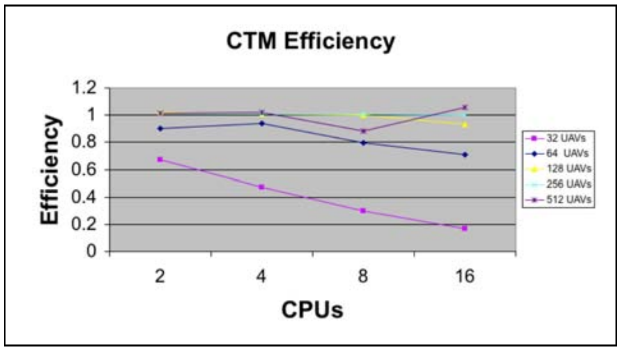
Regardless of philosophical arguments for optimistic time management in PDES, the results from Experiments 7 and 10 conclusively show for the given configurations and data sets that conservative time management with SPEEDES produces superior results using this particular swarm model. Consider the side-by-side comparison of runtimes given in Figure 6.7.



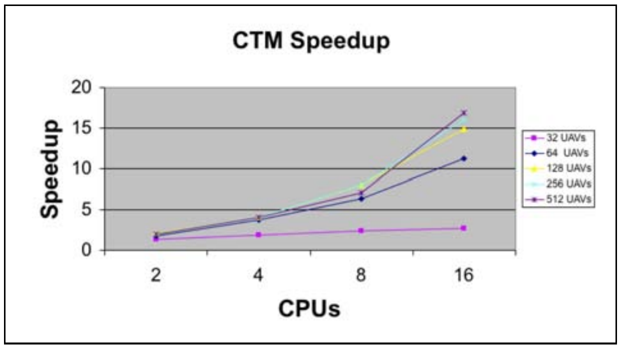
	1	2	4	8	16
CTM-256	4823.02	2385.35	1196.24	600.17	300.89
BTB-256	5268.34	4937.72	2451.41	996.29	739.81

Figure 6.7: Conservative time management outperforms the optimistic time management by approximately a factor of 2.

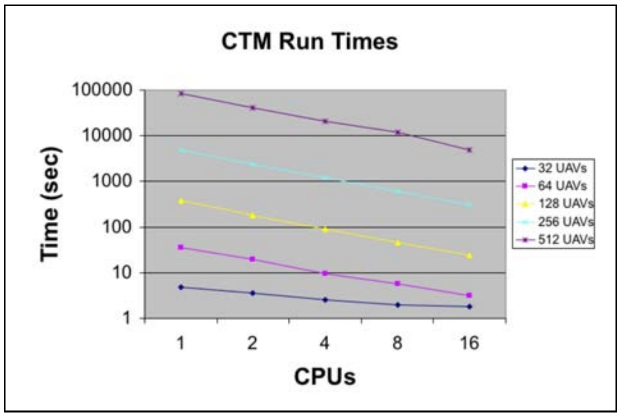
6.3.1 Synthesis of Experimental Results. Recall that the underlying objectives of the experiments in this suite were to produce an efficient and effective routing algorithm that can be synthesized into existing AFIT UAV swarm research and to provide parallel experimentation and analysis that can ultimately improve the performance of the AFIT UAV Swarm Simulator so that it can be of benefit to AFIT UAV research in general and to the sponsors of this research. The experiments in this suite more than satisfactorily realizes both of these objectives. An efficient and effective routing algorithm is produced that produces high quality solutions in very reasonable *a priori* computing times. In fact, the algorithm is fast enough to process dynamic routing changes while en route in addition to the *a priori* route scheduling.



(a) Efficiency



(b) Speedup



(c) Run Times

Figure 6.8: Results show that for a small swarm of UAVs, there is only negligible gain for parallelizing the swarm algorithm. For a larger swarm size, however, there is significant gain, but efficiency quickly becomes of predominant concern.

Additionally, sufficient analysis and data is produced that suggests a transition from an optimistic to a conservative time management scheme with the SPEEDES library improves simulation runtimes in the general case by more than factor of 2 in addition to providing much more ‘regular’ behavior with regard to parallel computing analysis. Such an enormous improvement is a large step toward simulating swarms of UAVs in almost realtime.

Given a routing algorithm, and a more high fidelity parallel simulation, the next logical step in the research cycle is to unify the existing routing algorithm and parallel simulation. This work can be accomplished by either replacing or modifying the existing swarm model from Kadrovach’s research [39] with a swarm model that can be routed amongst coordinates. While the existing swarm model does provide realistic swarm behavior, it is of little use without the ability to be routed. It is likely that the entire swarm model may need to be redesigned from the ground up if some of the guidelines from Kadrovach’s Ph.D dissertation are determined to not provide a satisfactory routing capability. *Routing a swarm should not be a two step process that can be decomposed into separate swarm behavior and routing models.* Rather, the two should be somewhat tightly coupled so that the decentralized control of the swarm can automatically accommodate changes and deviations from an original routing schedule. Once the simulation is extended to provide a swarm model that can be routed, the routing algorithm can provide an initial way point sequence for simulating sorties, and the simulation can then be extended to include threats, dynamic changes to the way point schedule, etc. One very high level architecture for a more unified simulation is shown in Figure 6.9

6.4 Summary

This chapter presented the results of experiments in measuring the efficiency and effectiveness of a particular routing algorithm employing a Genetic Algorithm. Results show that with the use of a repair function, the Schema Theorem holds, and very high quality results are rapidly approached

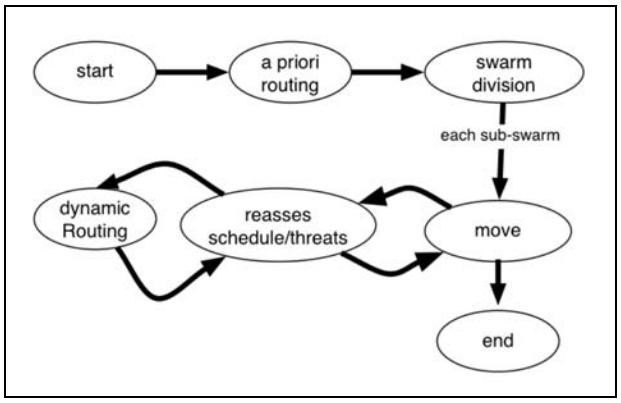


Figure 6.9: A high level swarm simulation architecture capable *a priori* and dynamic routing in addition to threat detection and other constraints.

as the result of good low-order buildling blocks exponentially increasing as the search progresses. This characteristic is desirable for a mission planning system.

Additionally, analysis from various CPU and swarm configurations simulated as a SPEEDES application shows that SPEEDES applications exhibit 'regular' parallel behavior with regard to speedup and efficiency when run in a conservative mode. Running in an optimistic mode, on the other hand, produces some abnormalities with the introduction of additional CPUs. The results for a UAV swarming application suggest that conservative time management outperforms optimistic techniques by approximately a factor of 2.

VII. Conclusions

A discovery is the product of a previous discovery
and in its turn will give rise to a further discovery
–Joseph Louis Gay-Lussac

THIS research investigation begins by presenting Unmanned Aerial Vehicle (UAV) research as the ultimate scientific engineering endeavor and by establishing the principal motivation of this particular UAV research as the desire to empower our nation's military to fight smarter battles and consequently save more lives. Routing and simulating large swarms of UAVs are two pillars directly supporting UAV research and are the overarching concepts addressed in this particular research investigation. The remainder of this chapter reviews the results of this endeavor and the completion of its objectives.

7.1 Completion of Objectives

Objective 1: Formulate, implement, and empirically measure the performance of a swarm routing algorithm by means of efficiency and effectiveness metrics

The analysis and discussion related to Experiments 1-6 as presented in Chapters 5 and 6 complete this objective. A robust routing algorithm employing a Genetic Algorithm is developed that produces high quality solutions to hard Vehicle Routing Problem instantiations in very little time. This algorithm is shown to be efficient, to be able to adapt to a changing search landscape, and to produce optimal or near optimal results for common benchmark problems.

Objective 2: Analyze, evaluate, improve the efficiency of, and extend the capabilities of the existing AFIT Swarm Simulator as a Parallel Discrete Event Simulation and a swarm intelligence model

The entire experimentation suite from Chapters 5 and 6 support this objective. Furthermore, an additional discussion on synthesizing the routing algorithm into a parallel swarm simulation based on previous AFIT research using the Synchronous Environment for Emulation and Discrete

Event Simulation (SPEEDES) based upon the results from the experimentation suite supports this objective's fulfillment.

Objective 3: Application and in depth analysis of a Genetic Algorithm for online UAV routing Analysis from Experiments 7-10 as presented in Chapters 5 & 6 directly fulfill this objective. Results show that previous research using an optimistic time management method for communication between members of a swarm yields inferior results to a conservative time management scheme. Although somewhat unexpected, speedup is even superlinear in some of the test cases.

7.2 Research Contributions

Previous research showed the feasibility of modeling and visualizing swarm dynamics for use in UAV research, and an extension of this work resulted in a working parallel implementation. A working parallel implementation of swarming reconnaissance is a significant accomplishment, but several important aspects of functionality still remained: 1) A system to provide an efficient and effective routing schedule for the swarm to traverse, and 2) The need to remove overwhelming inefficiencies from the parallel simulation so that realistic UAV sorties can be accomplished and visualized in reasonable lengths of time.

This research investigation fulfills both of these functional aspects through a suite of relevant experimentation and analysis. Additionally, this effort provides a more stable simulation test bed via the refactoring of selected software components and providing a friendlier and extensible user interface to the simulation.

The simulation results from this investigation that show an almost superlinear speedup resulting from a switch to a conservative time management scheme for the parallel simulation provide a pivotal step toward simulation, routing, and visualizing large swarms of UAVs in direct support of the Air Force Research Laboratory Electronic Warfare Sensors Directorate (AFRL/SNZW), and in particular, its Virtual Combat Laboratory. Ultimately, this research and its impact on

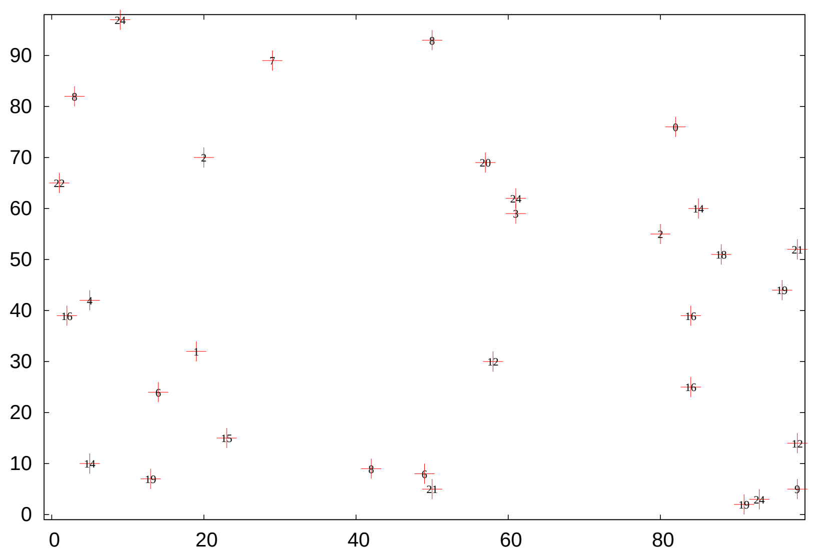
AFRL/SNZW means being able to fight smarter and more effective battles, and saving human lives by keeping them out of harm's way.

7.3 Future Work

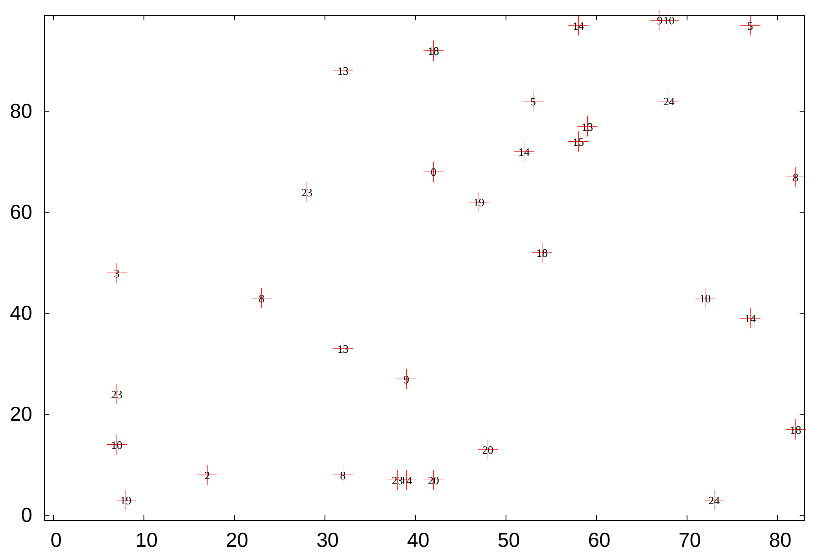
Future work should refactor, extend, or provide an alternate swarm model that is capable of starting and ending at desired coordinate locations. Without the capability, a swarm model–no matter how realistic or efficient–is of little use to real world applications. The new or enhanced model of swarm behavior should carefully consider the effects of routing individual swarm members on the effects of overall swarm dynamics. For use in a parallel simulation, it is crucial that the swarm routing algorithm be decomposable into constituent subparts that lend themselves to parallel computation. Given a routable swarm model, the routing algorithm as developed in this research can be fully integrated and extended to accommodate a variety of constraints.

In order to simulate much larger swarms of UAVs, the simulation testbed should be configured to run on a larger Beowulf cluster involving more than the 32 processors currently available at the time of this work. Additional configuration and customization of the SPEEDES simulation settings is also likely to provide additional performance enhancements, because SPEEDES can be finely tuned for specific applications. Research as this point has not attempted to eliminate unnecessary inefficiencies in the protocol stack or to fine tune many of the parameter values available in SPEEDES.

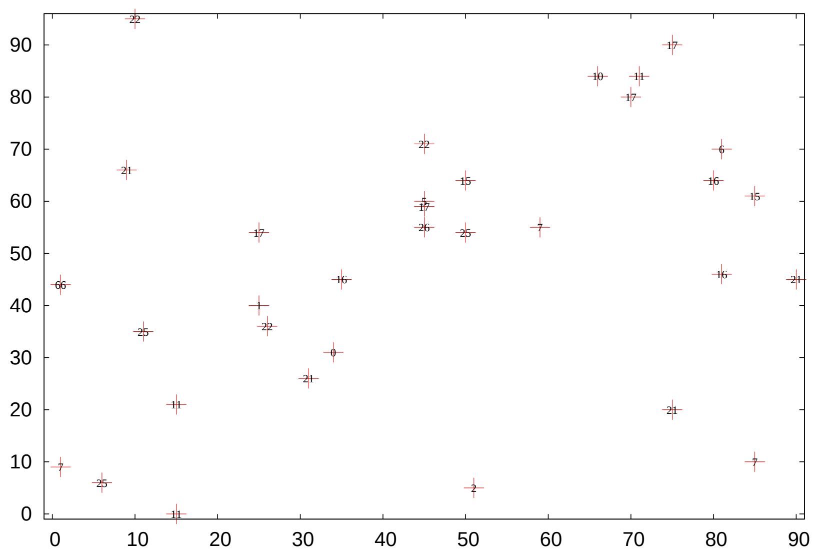
A-n32-k5



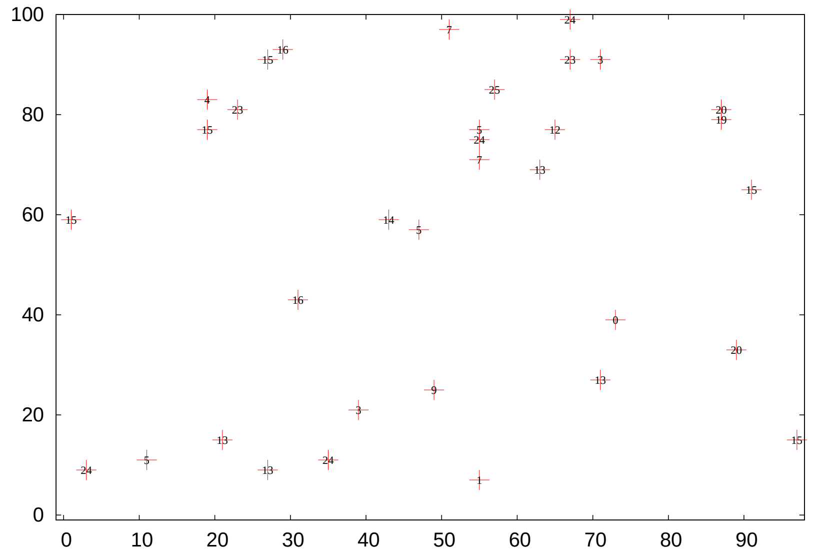
A-n33-k5



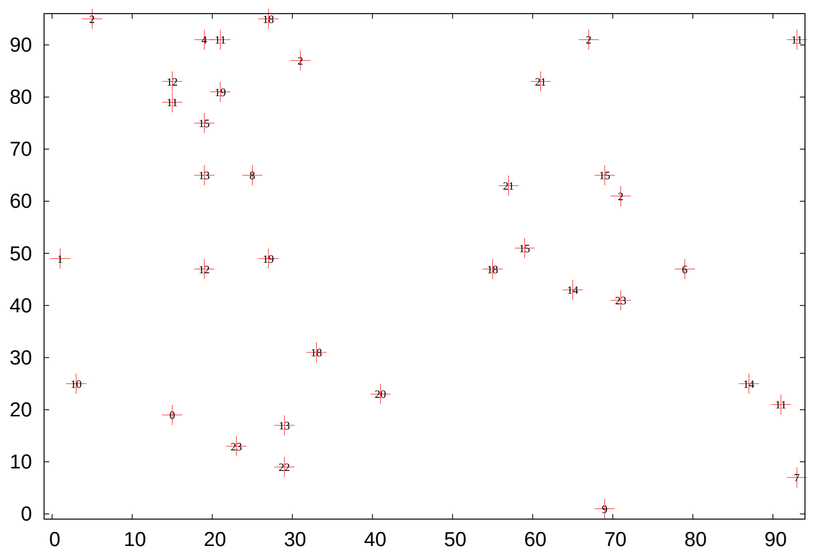
A-n33-k6



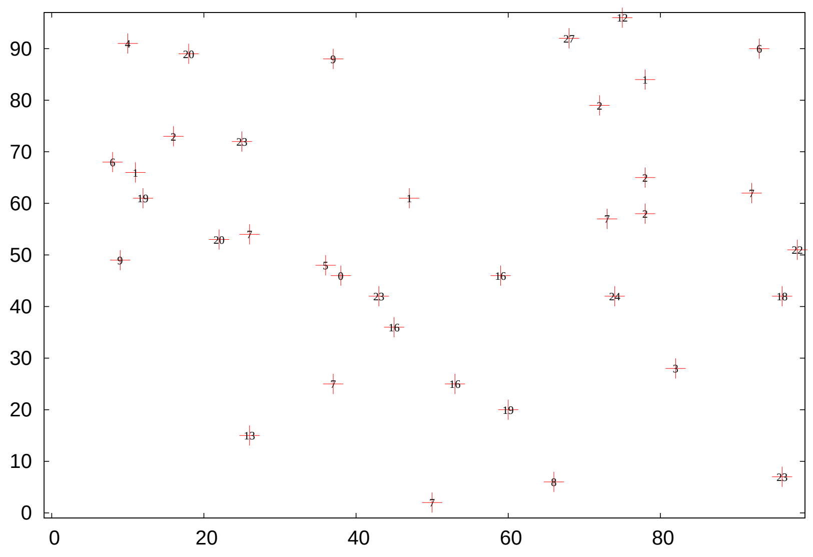
A-n34-k5



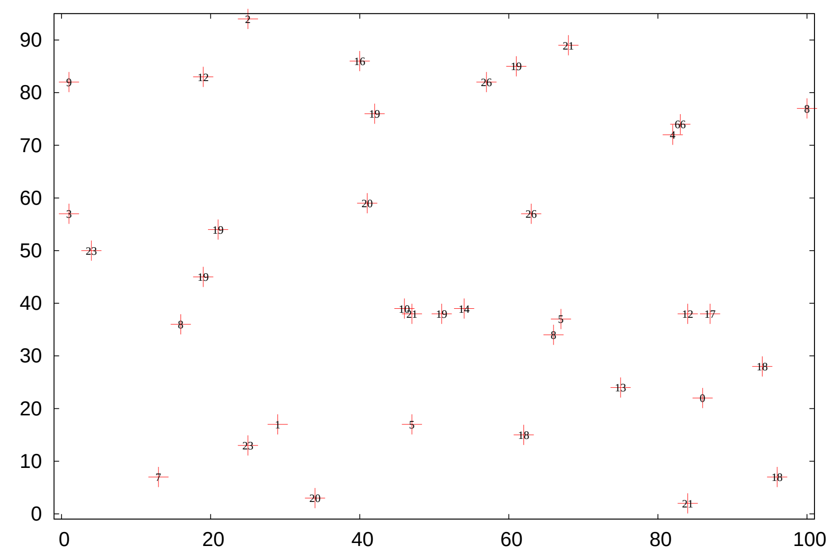
A-n36-k5



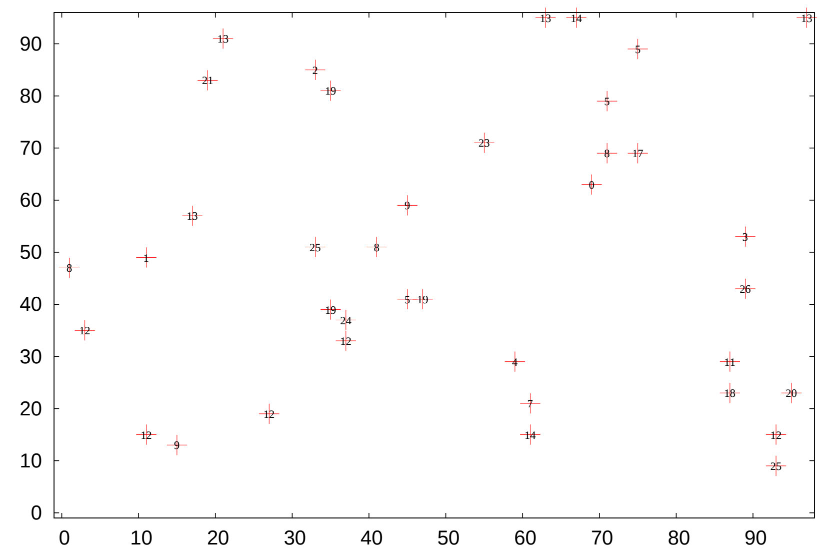
A-n37-k5

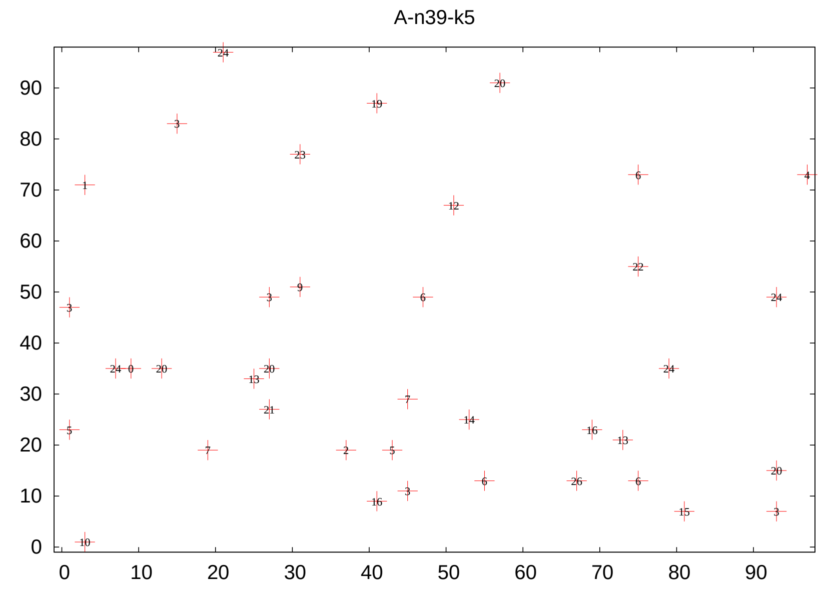


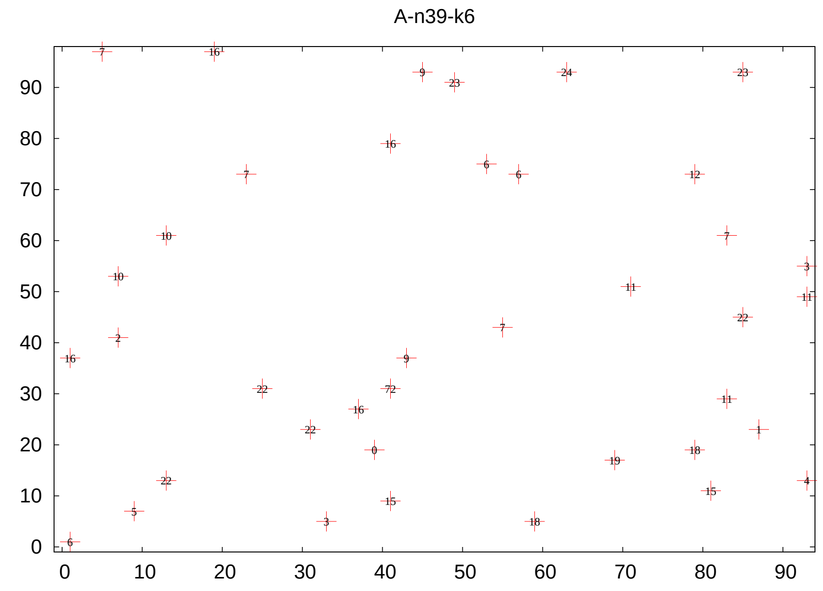
A-n37-k6



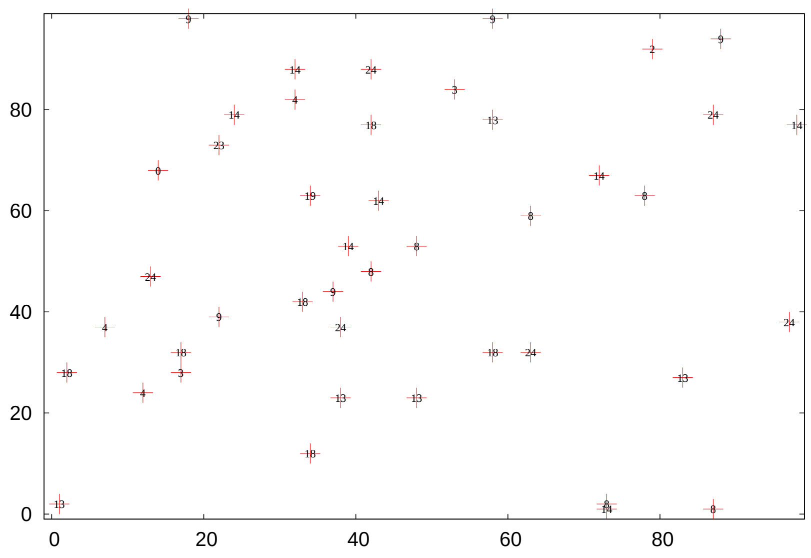
A-n38-k5



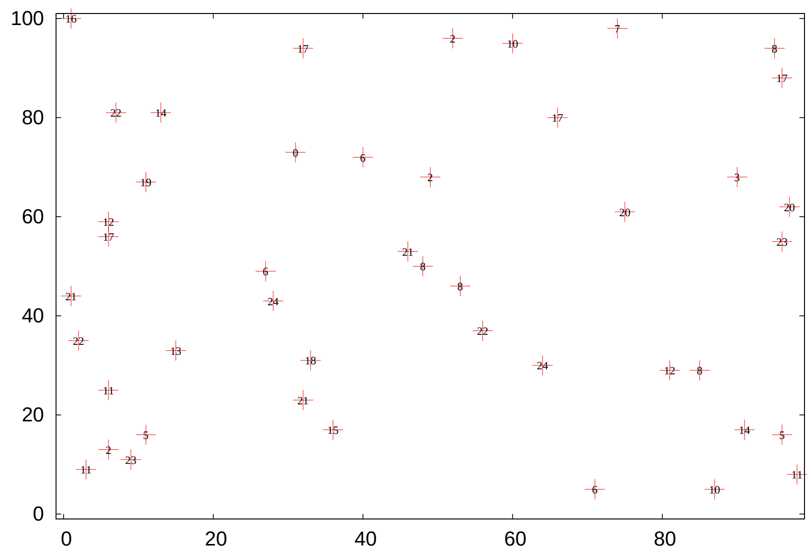

A-n39-k5


A-n39-k6

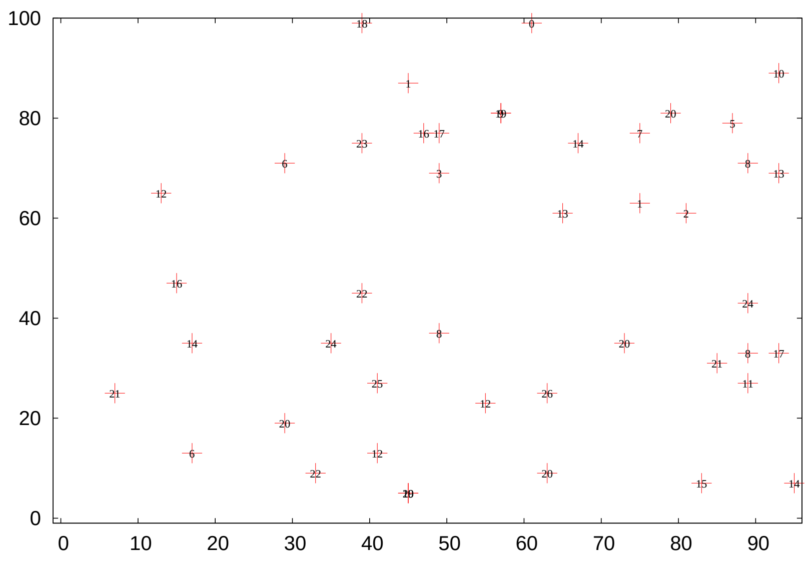
A-n44-k6



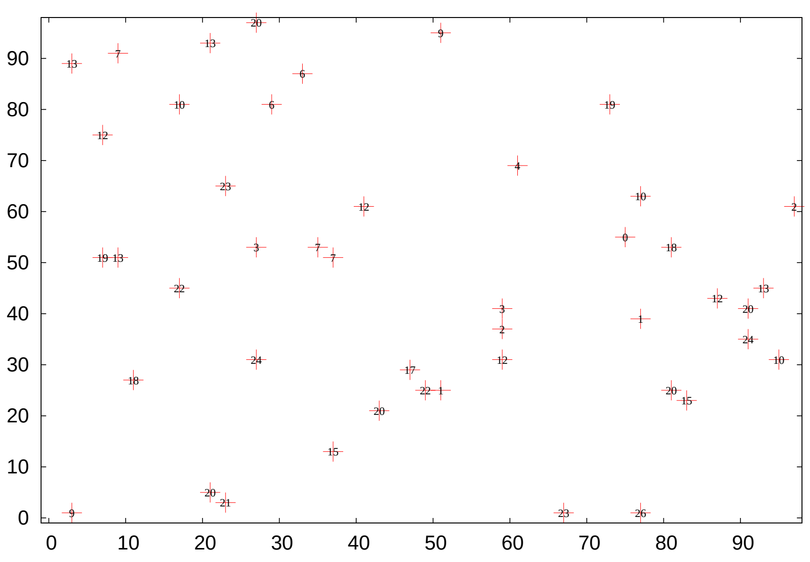
A-n45-k6

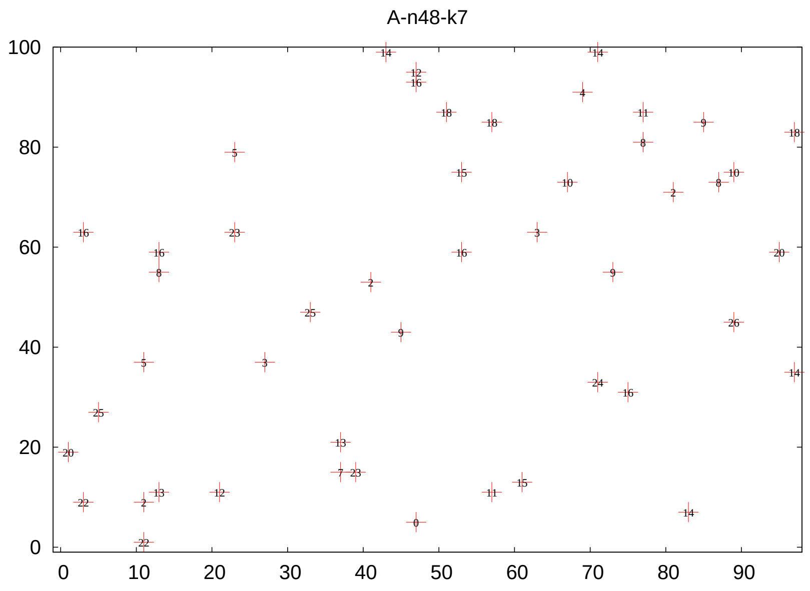


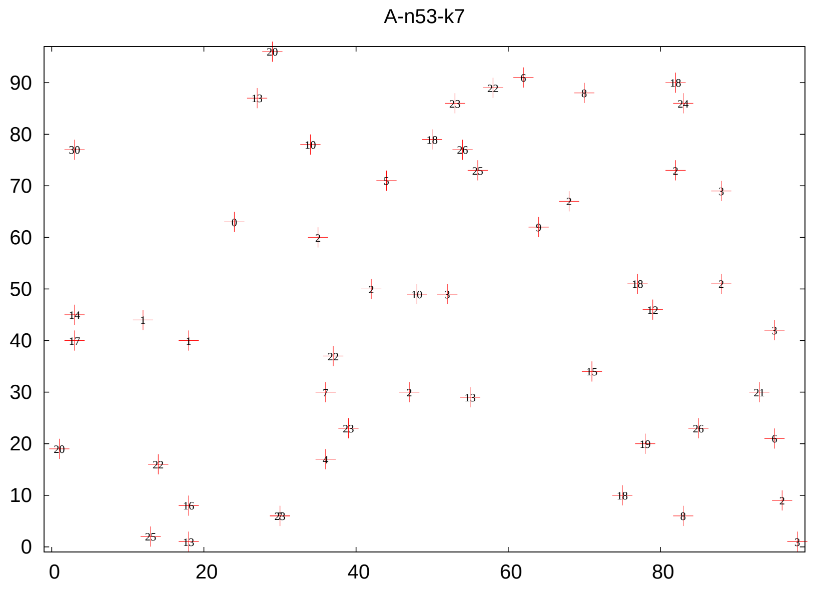
A-n45-k7

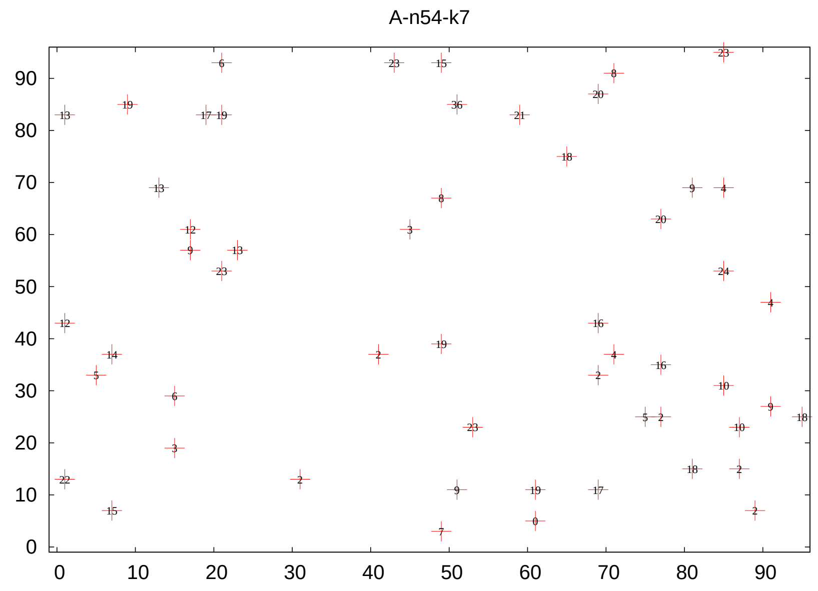


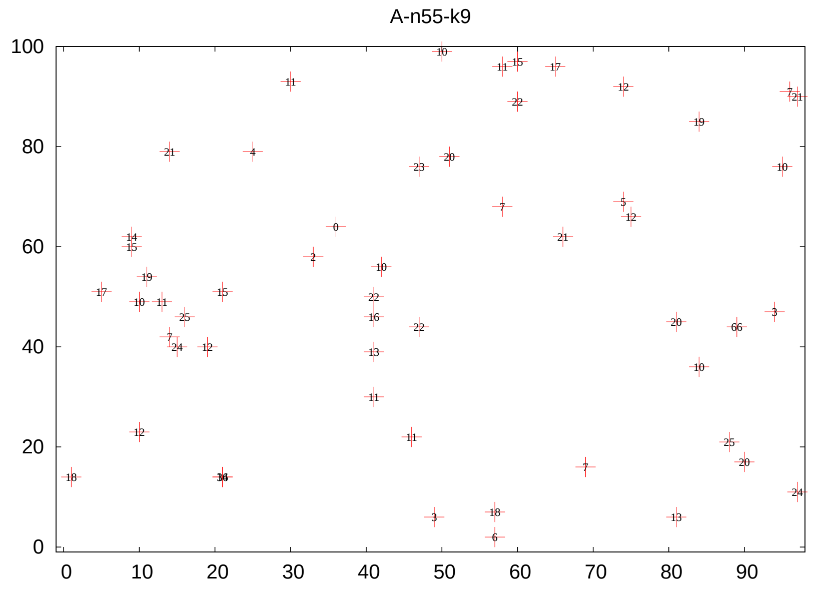
A-n46-k7



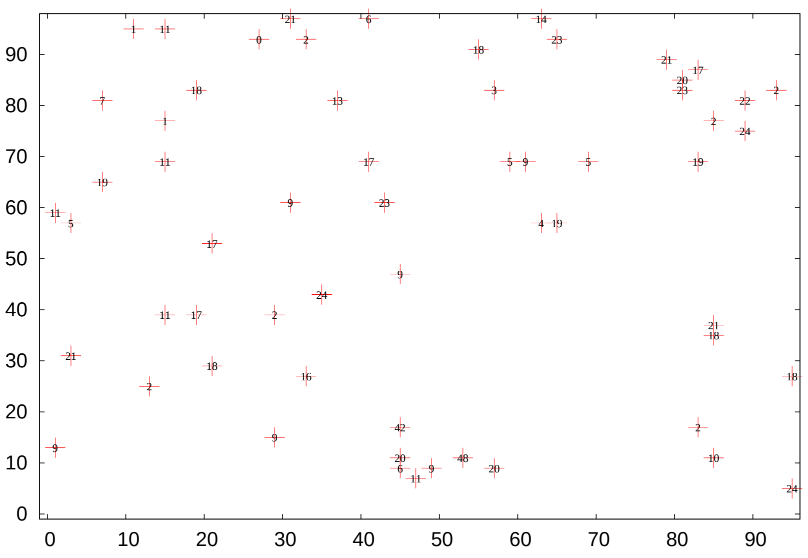




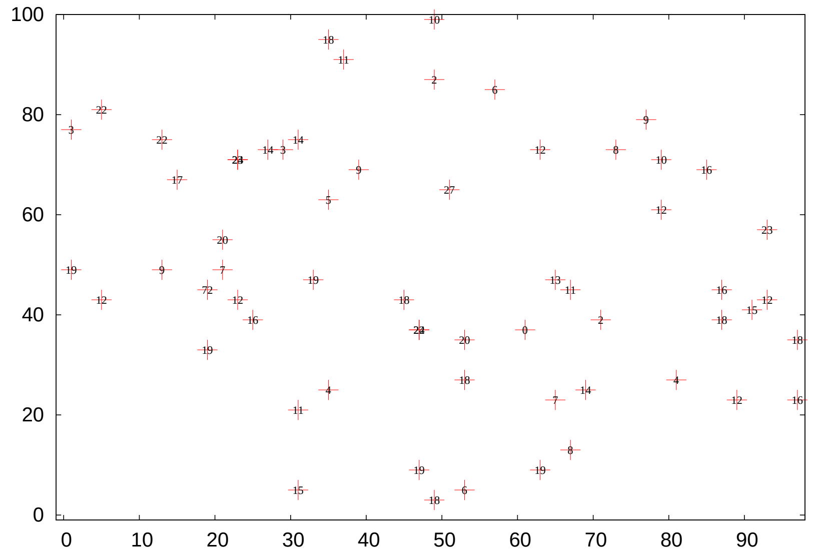

A-n54-k7


A-n55-k9

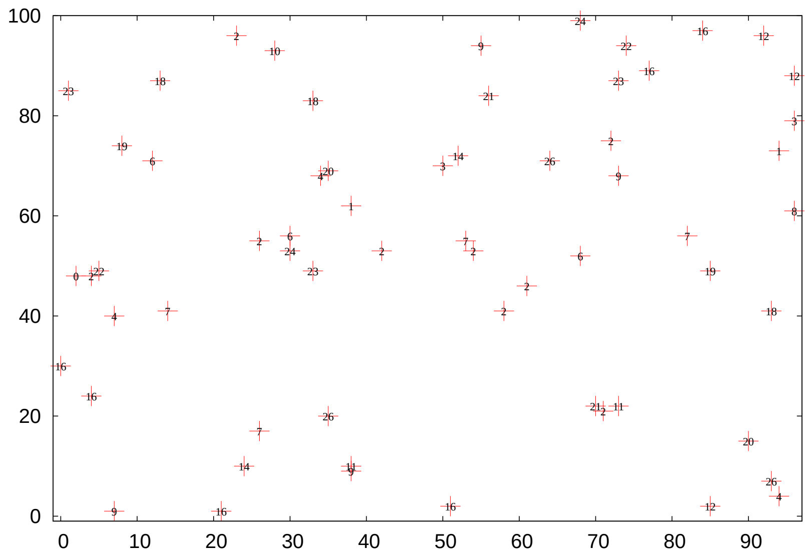
A-n60-k9



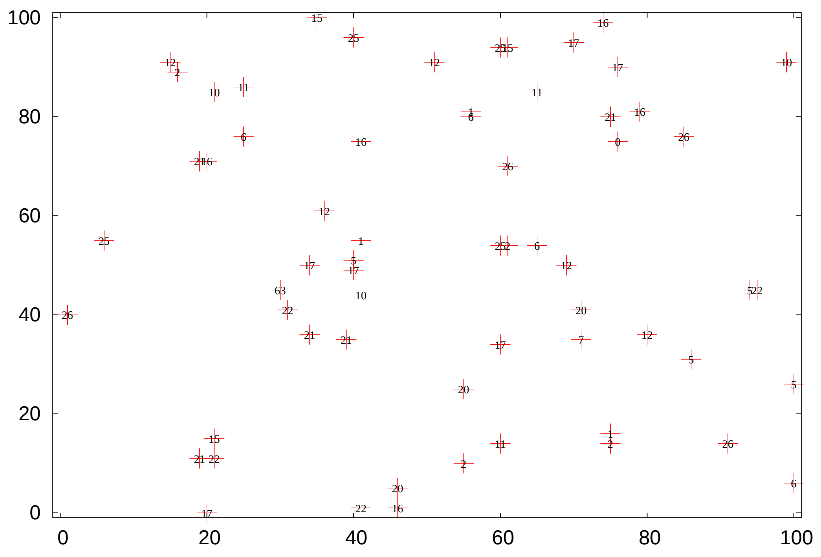
A-n61-k9



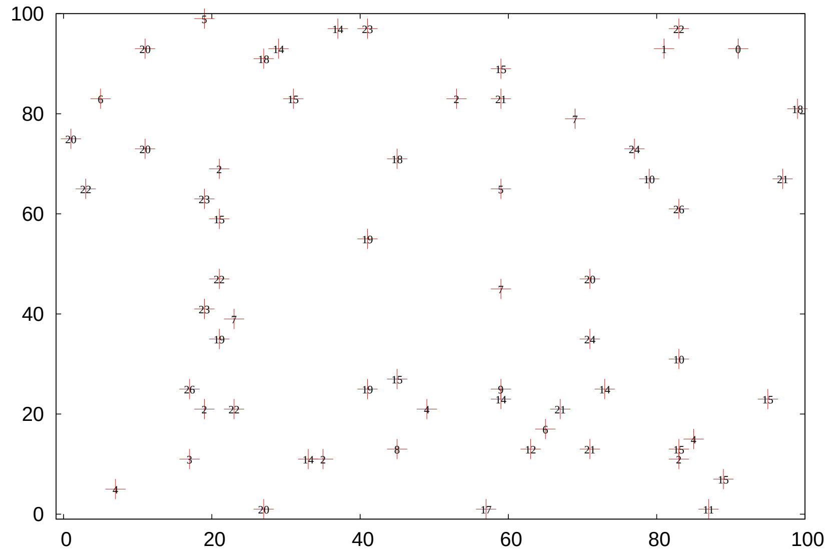
A-n62-k8



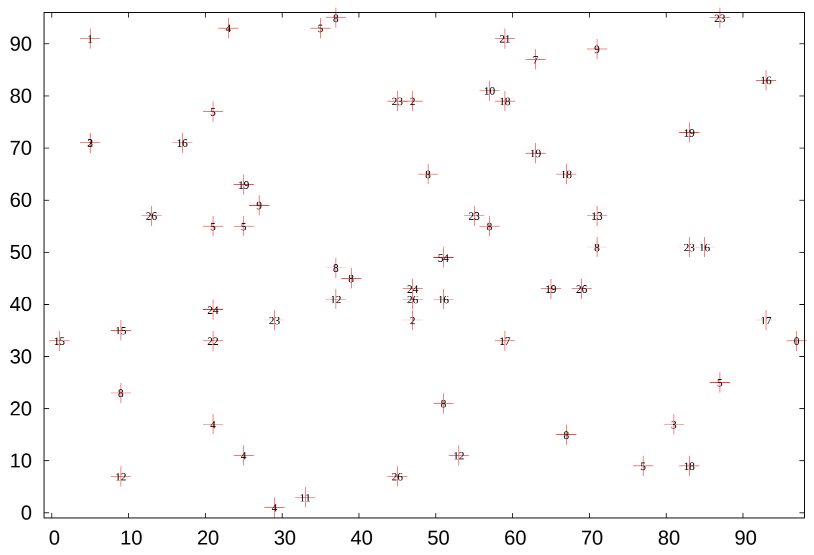
A-n63-k10



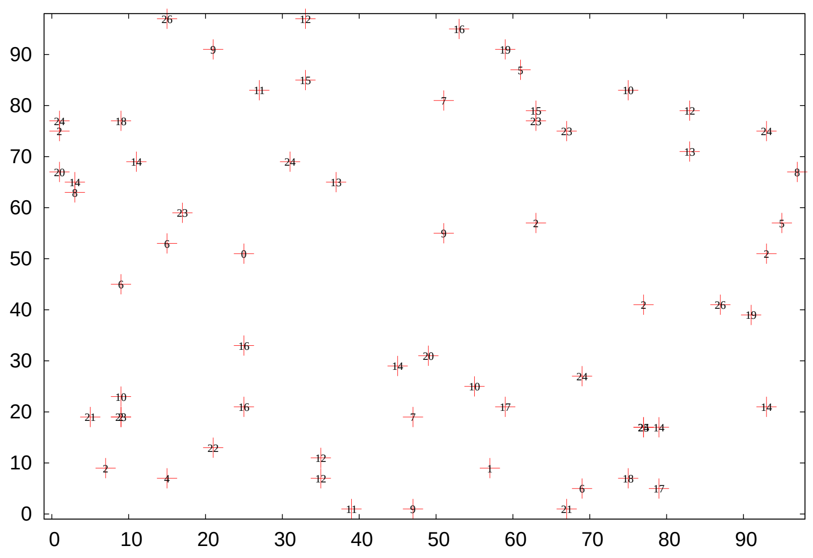
A-n63-k9



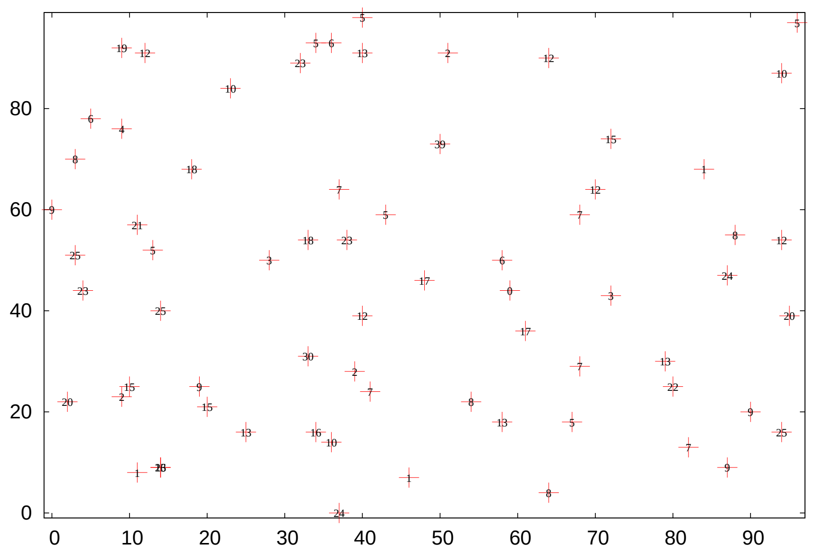
A-n64-k9



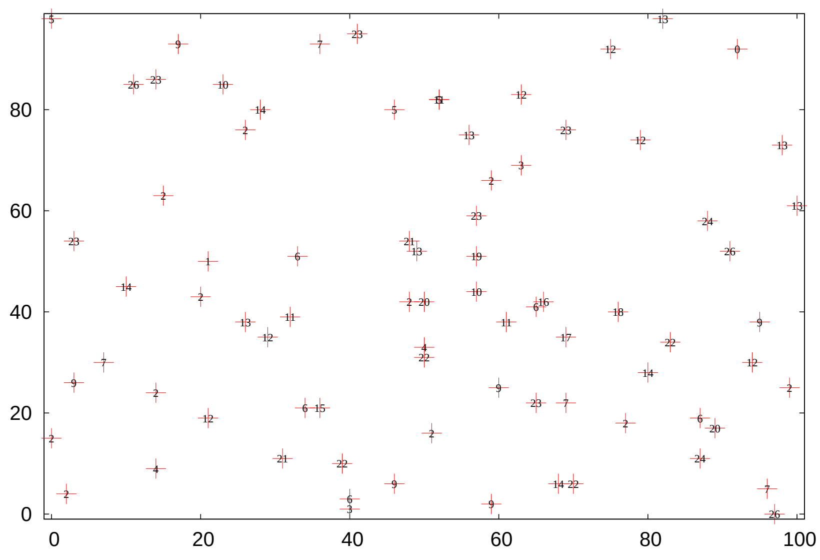
A-n65-k9



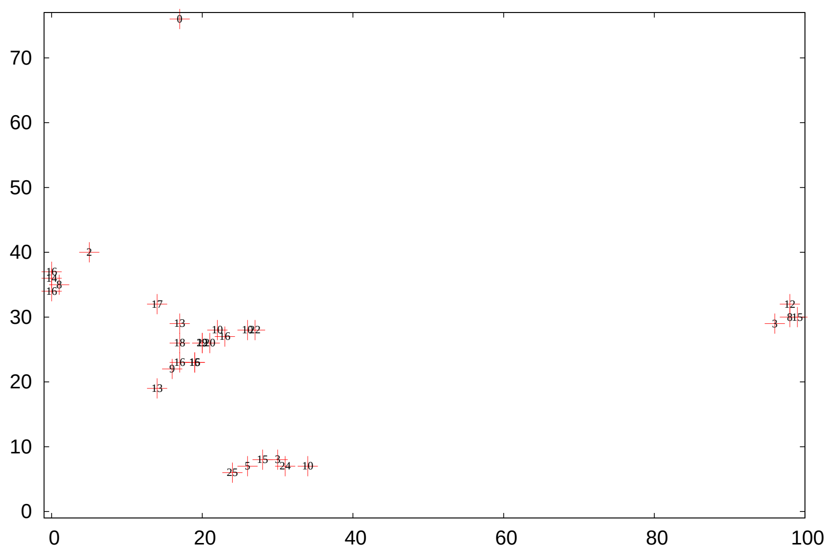
A-n69-k9



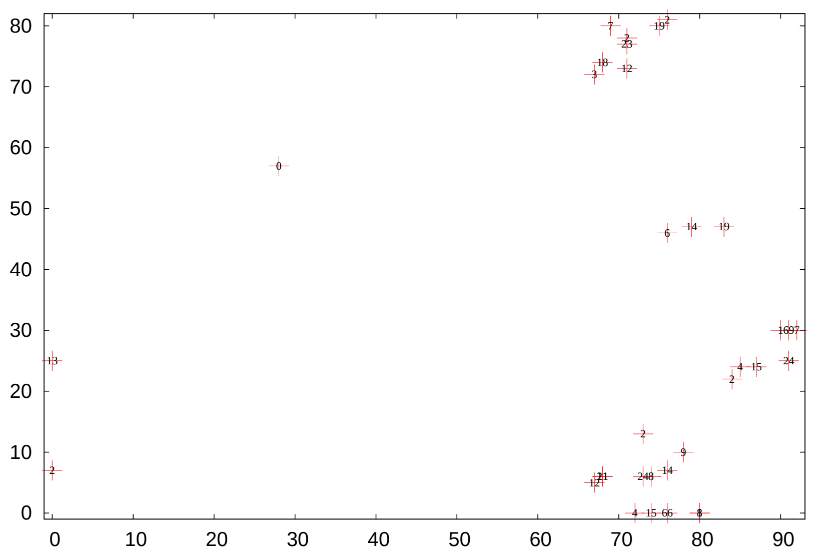
A-n80-k10



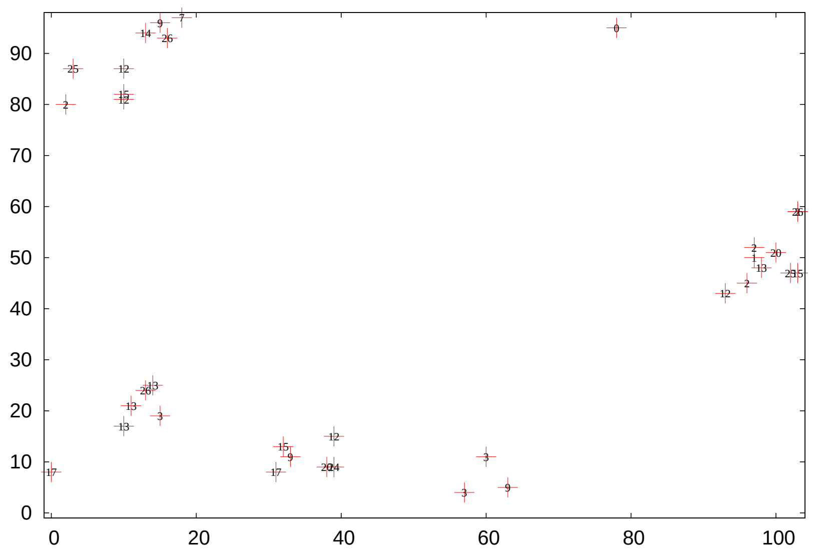
B-n31-k5



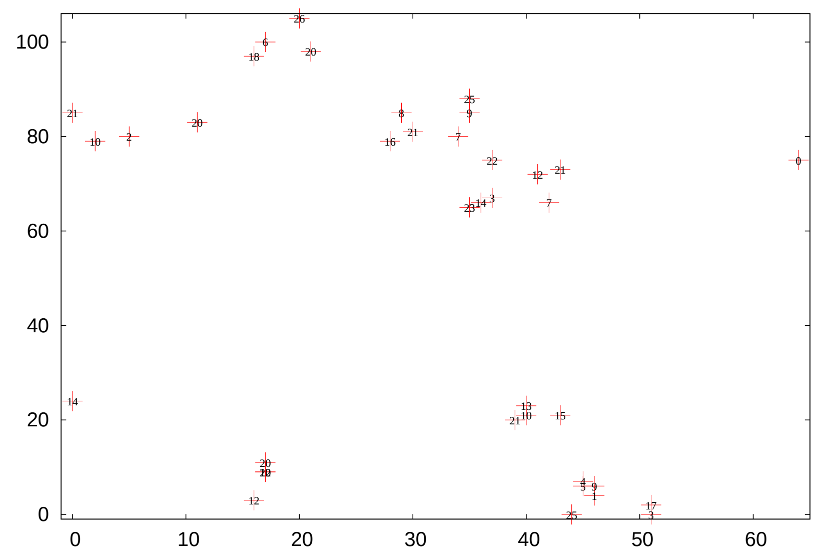
B-n34-k5



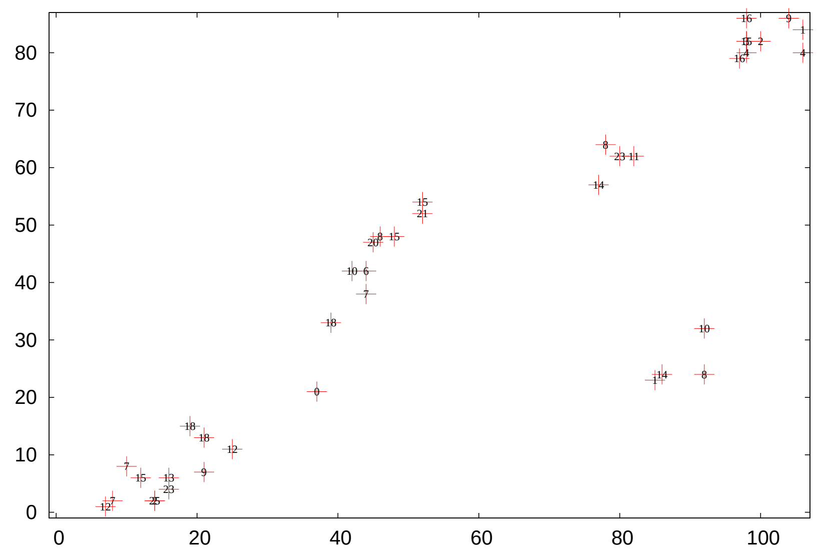
B-n35-k5



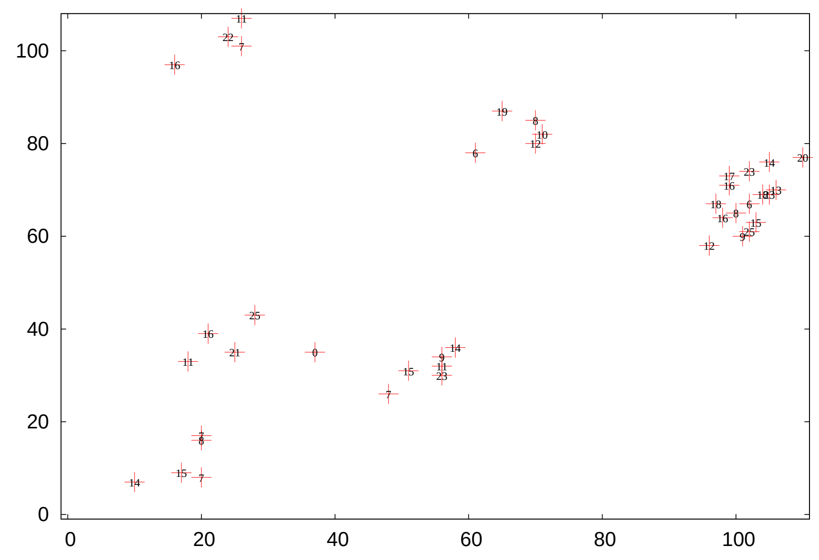
B-n38-k6



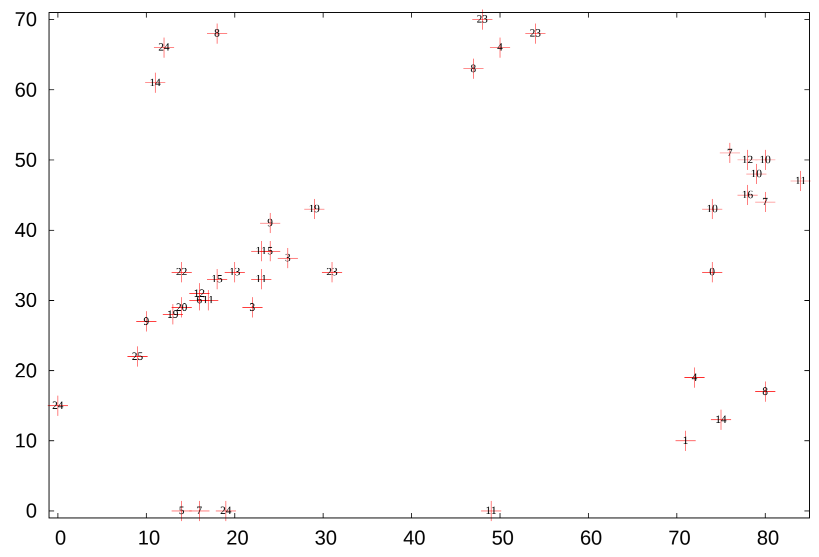
B-n39-k5



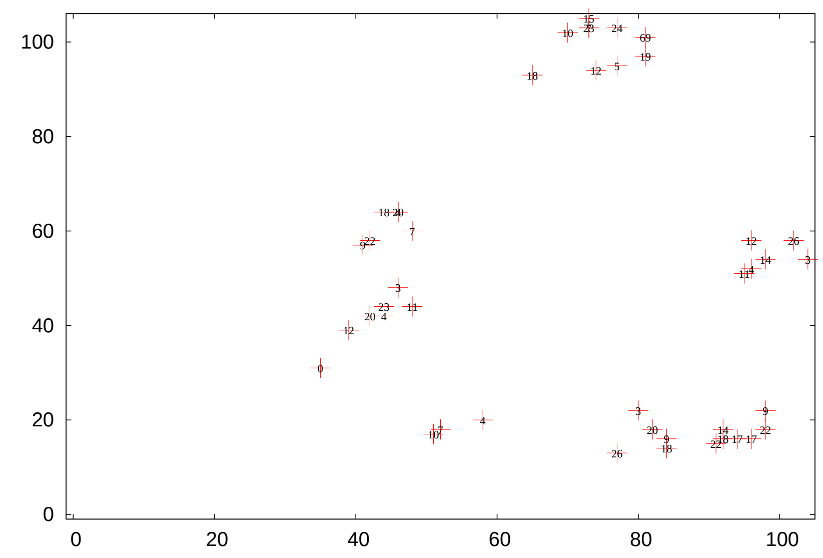
B-n41-k6



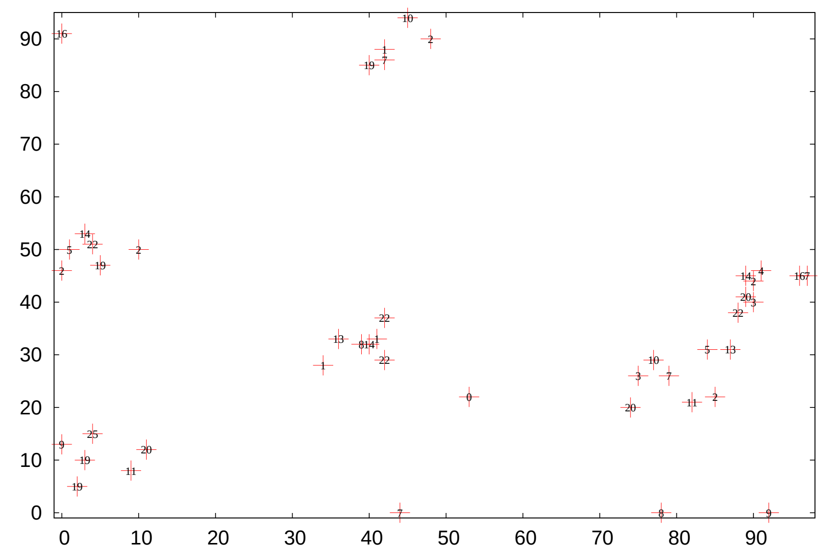
B-n43-k6



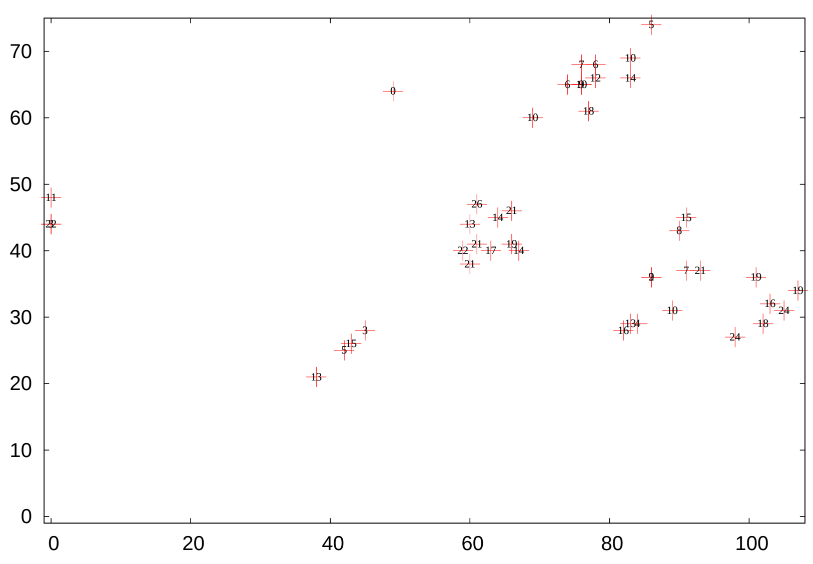
B-n44-k7



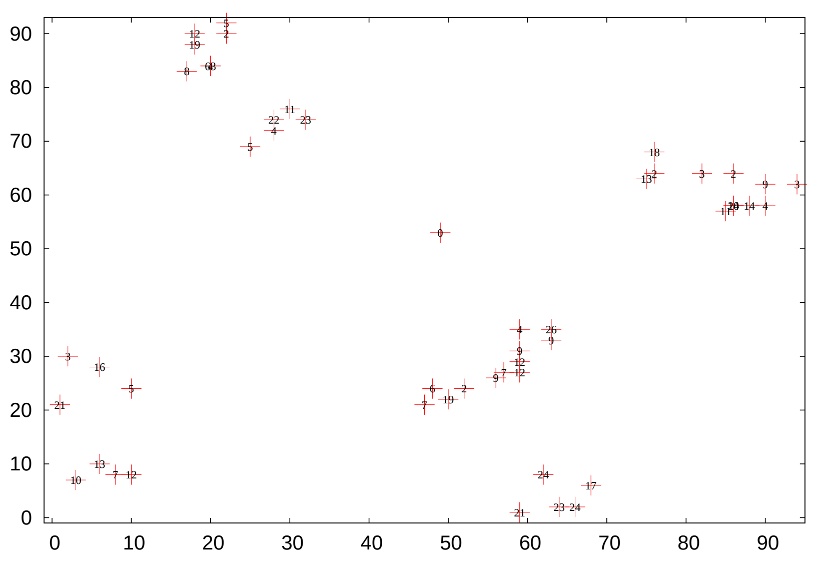
B-n45-k5



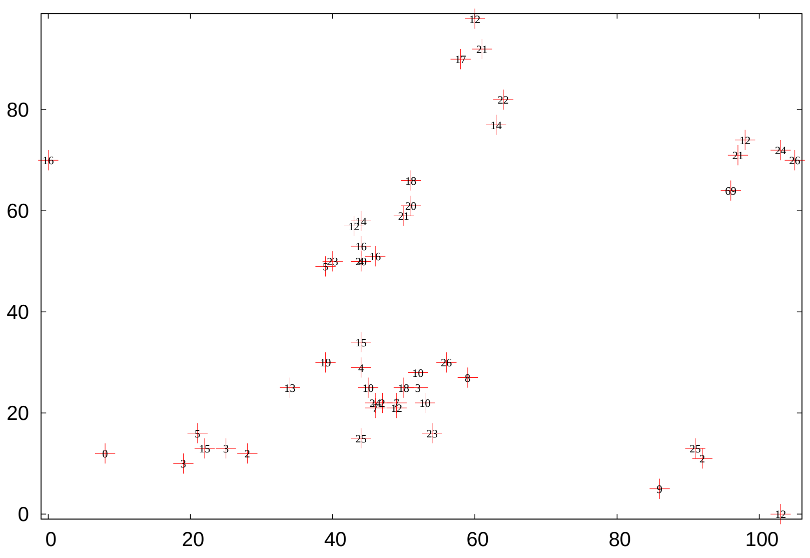
B-n45-k6



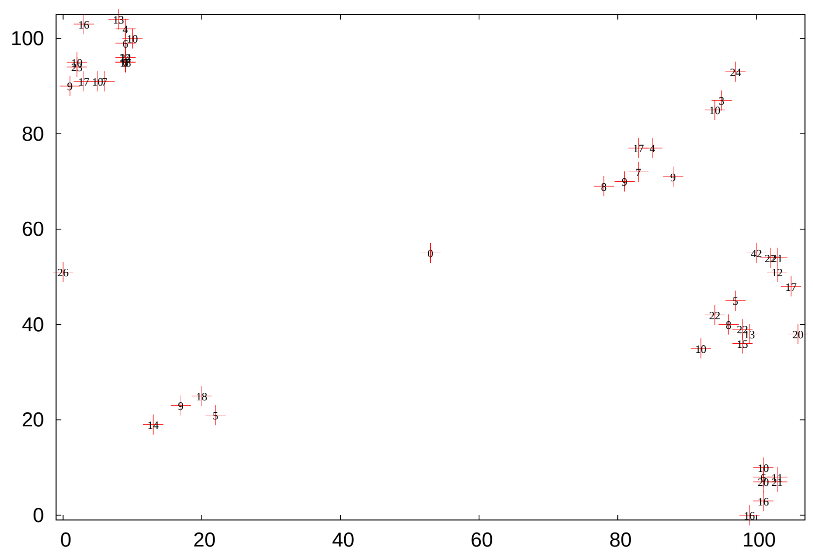
B-n50-k7



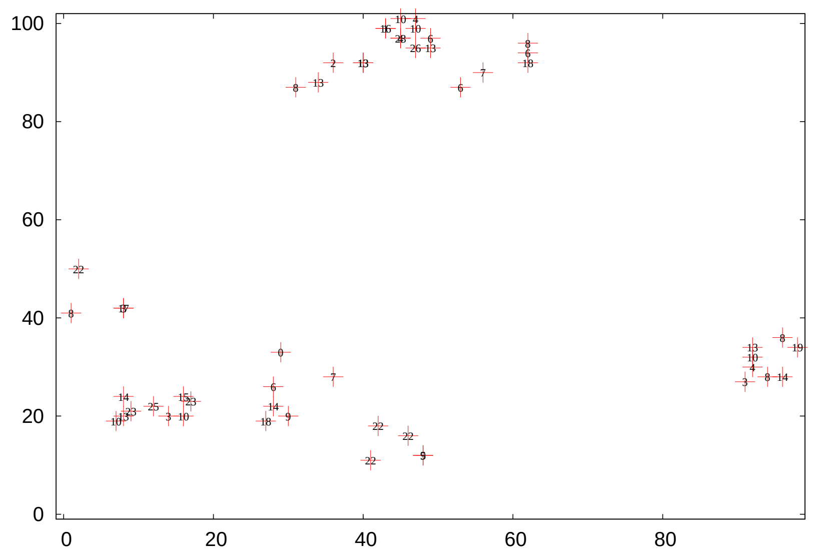
B-n50-k8



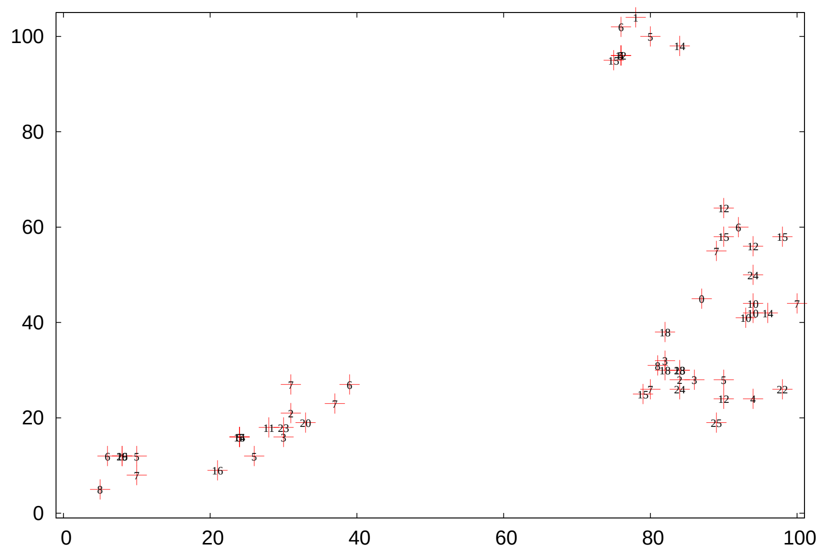
B-n51-k7



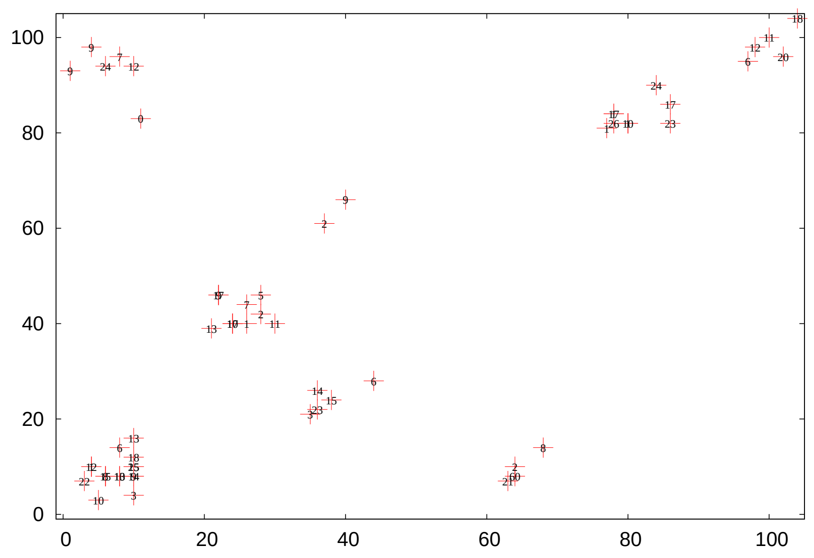
B-n52-k7



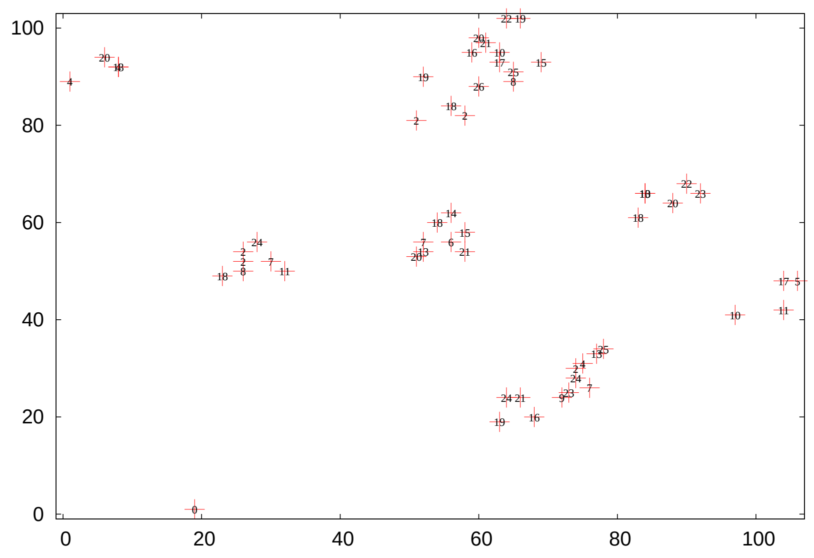
B-n56-k7



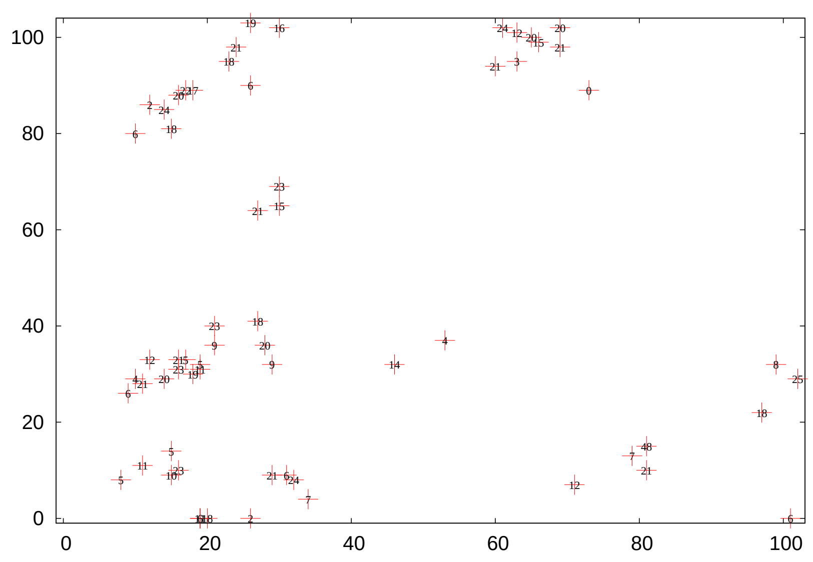
B-n57-k7



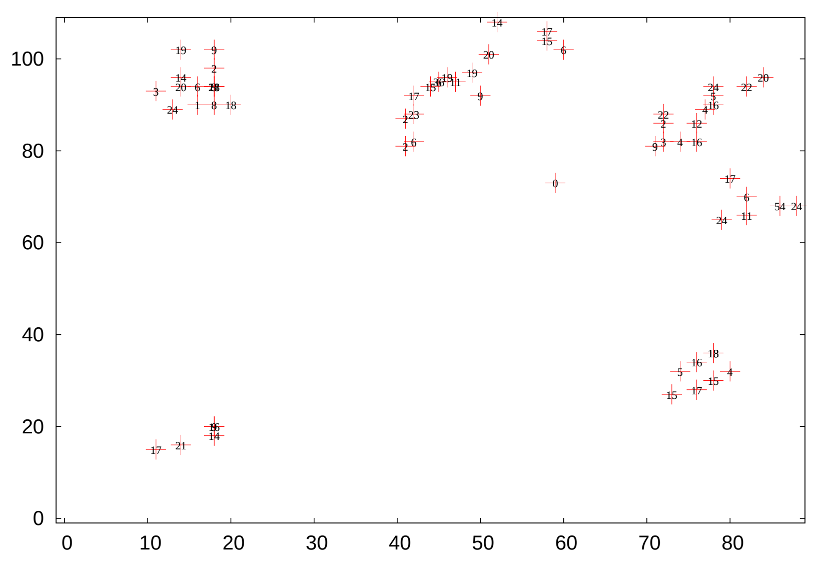
B-n57-k9



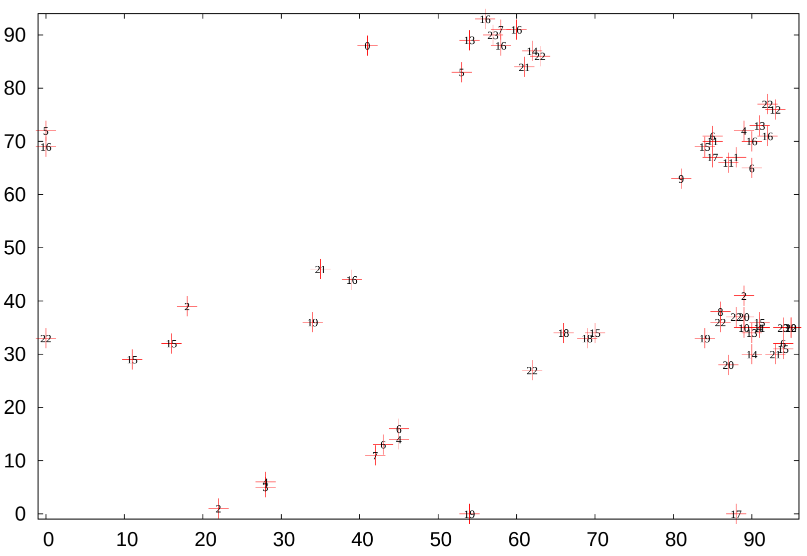
B-n63-k10



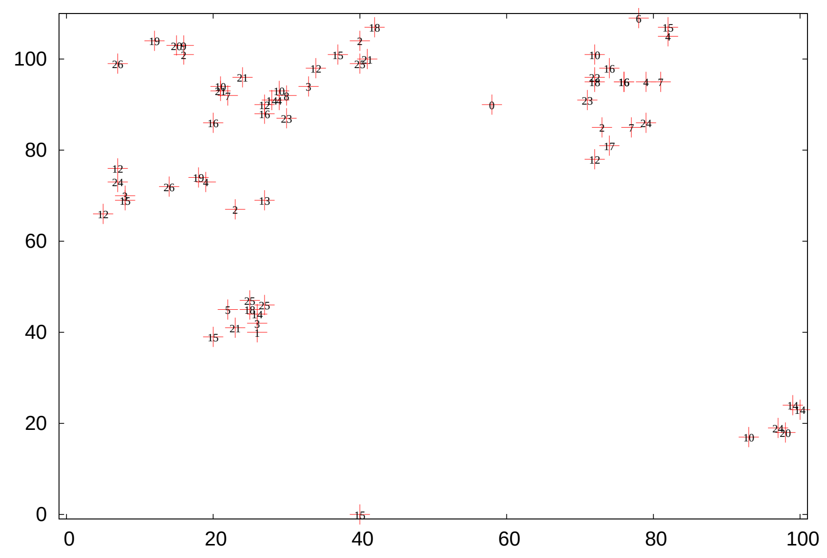
B-n64-k9



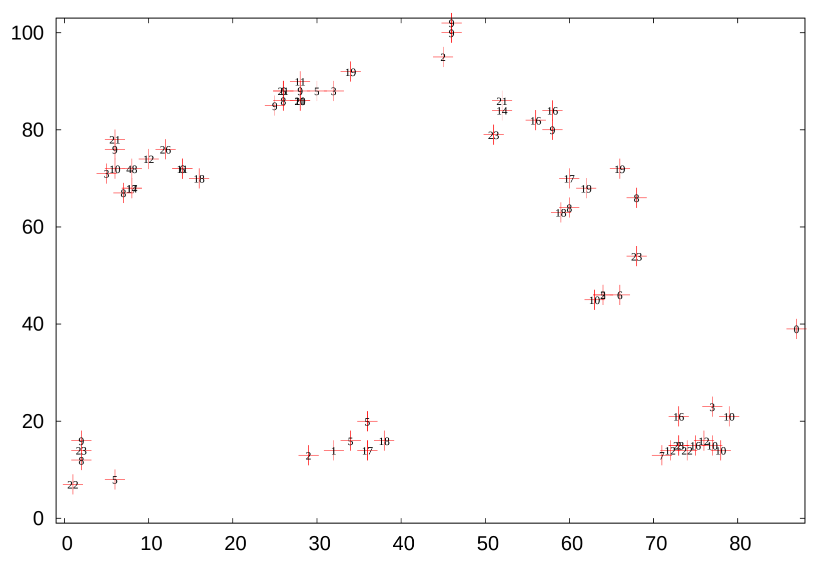
B-n66-k9



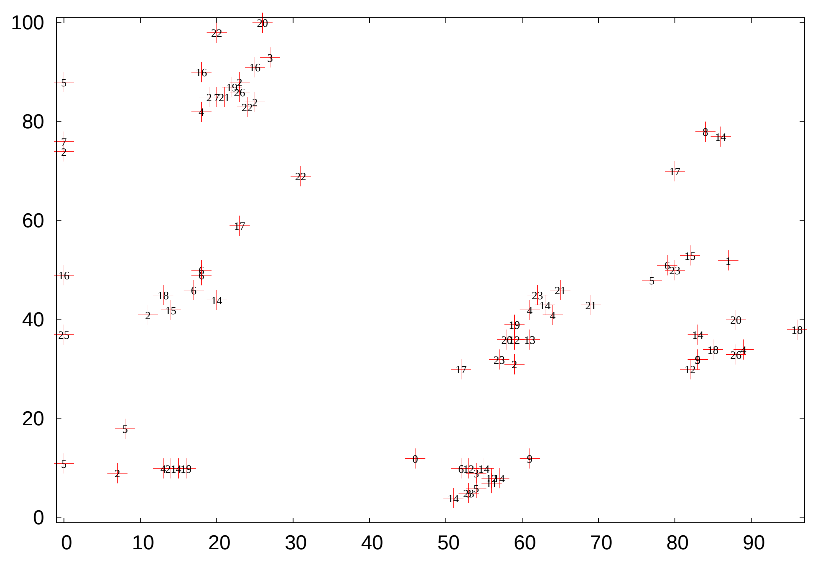
B-n67-k10



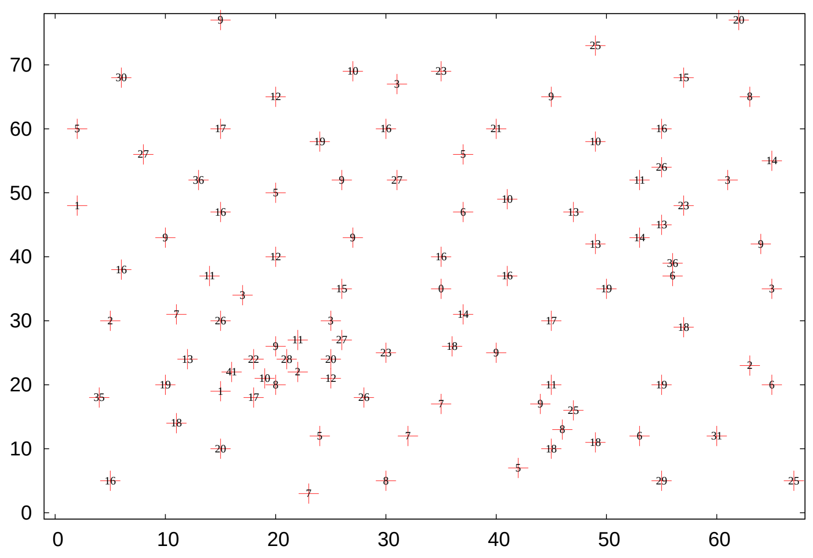
B-n68-k9



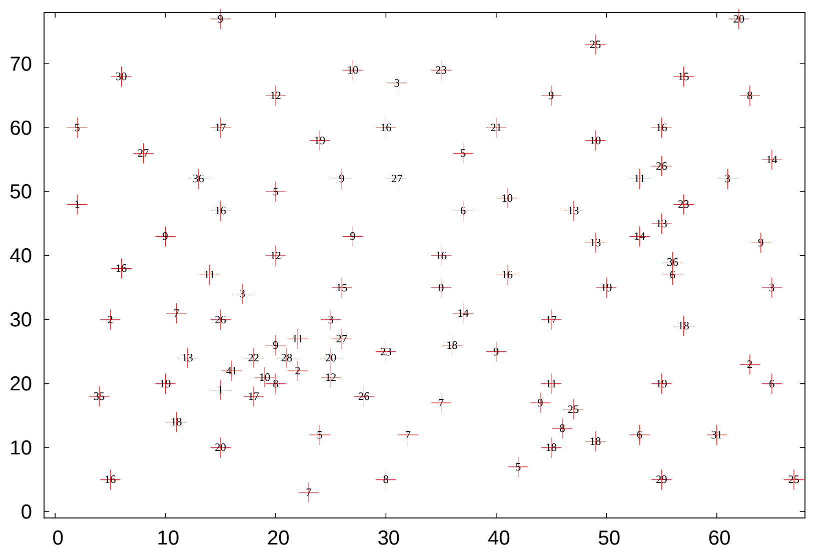
B-n78-k10

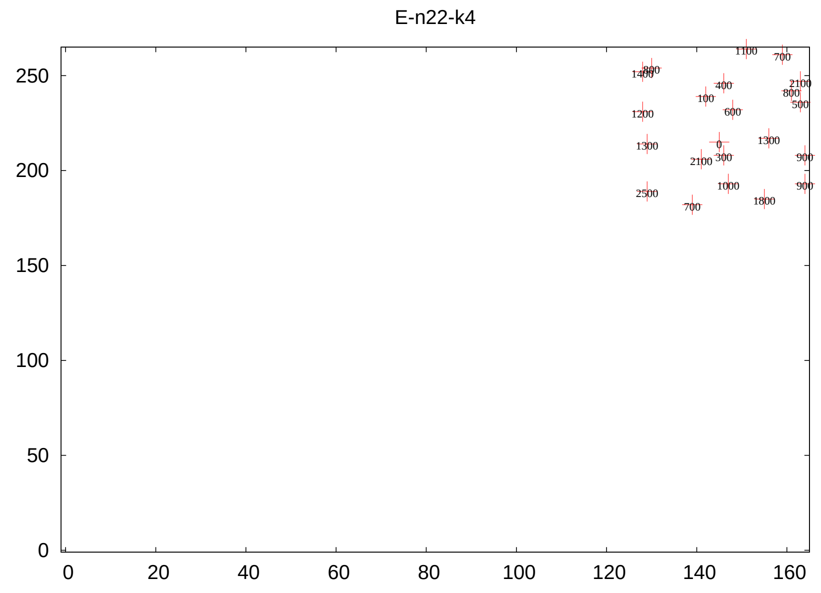


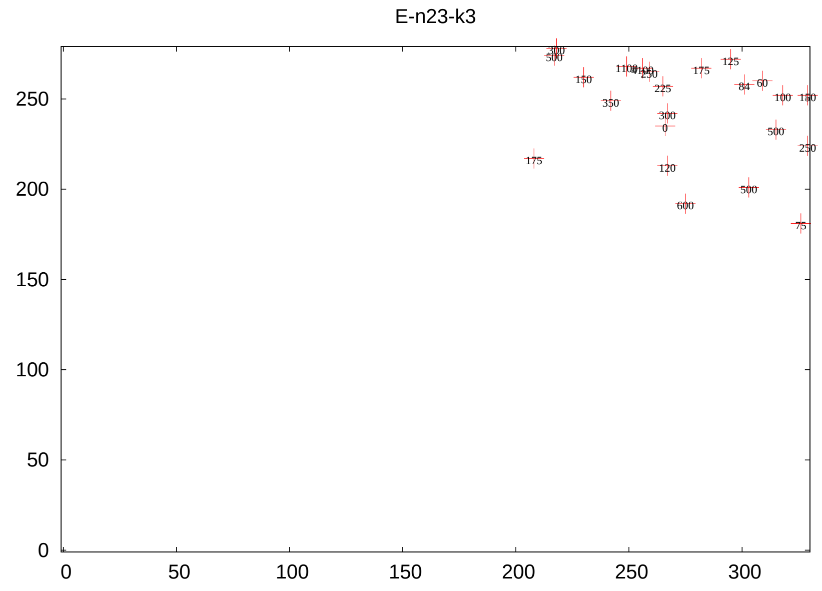
E-n101-k14

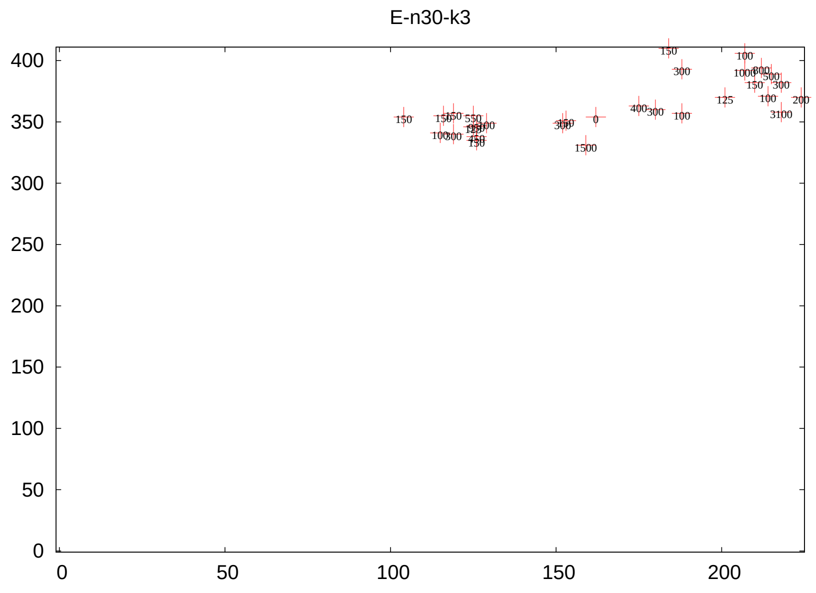


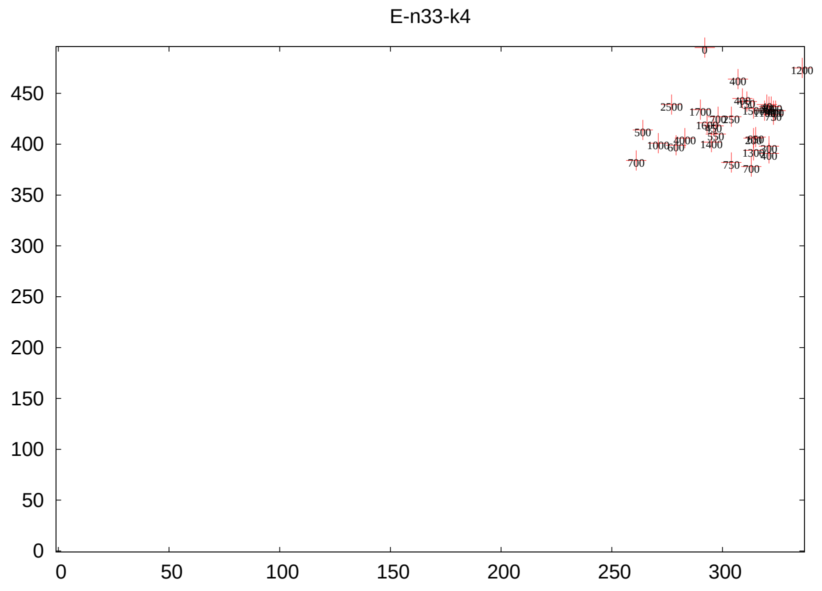
E-n101-k8

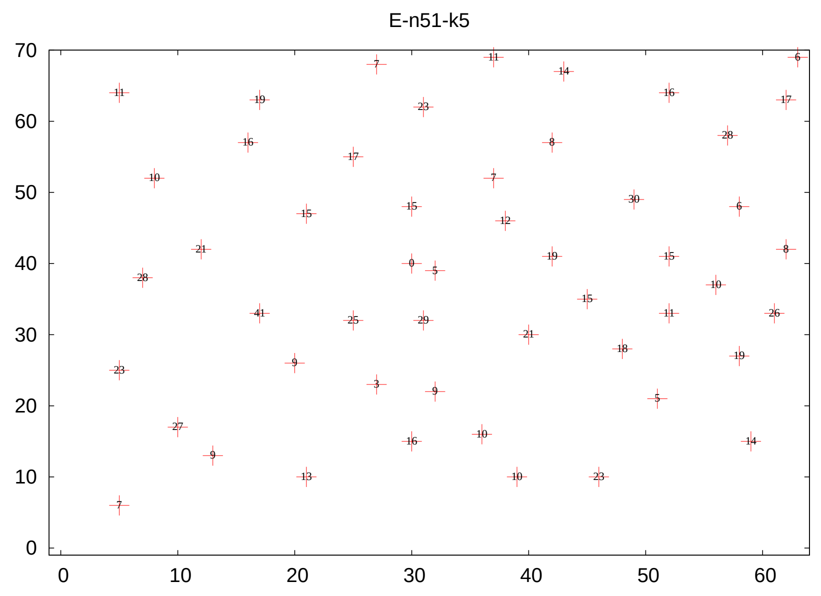


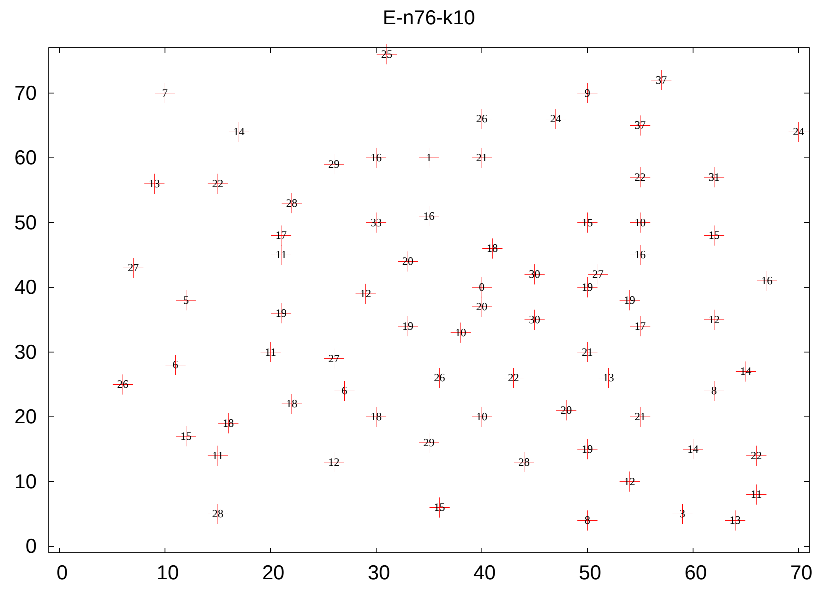

E-n22-k4


E-n23-k3

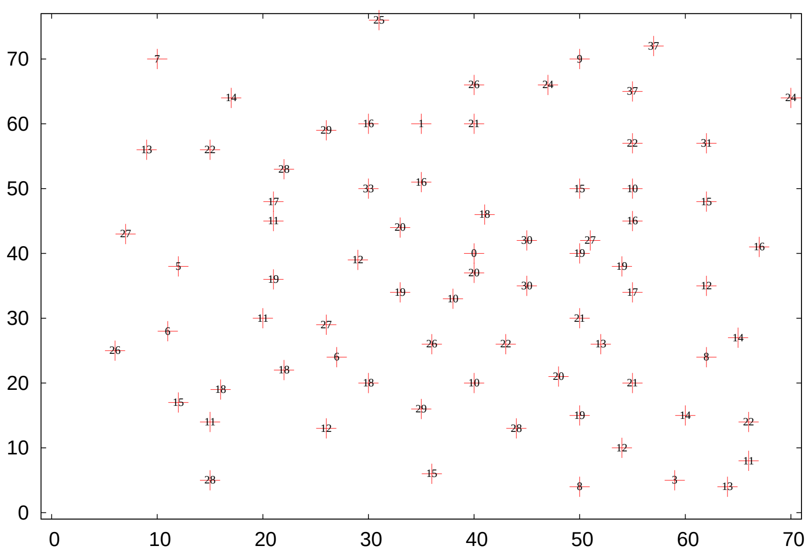




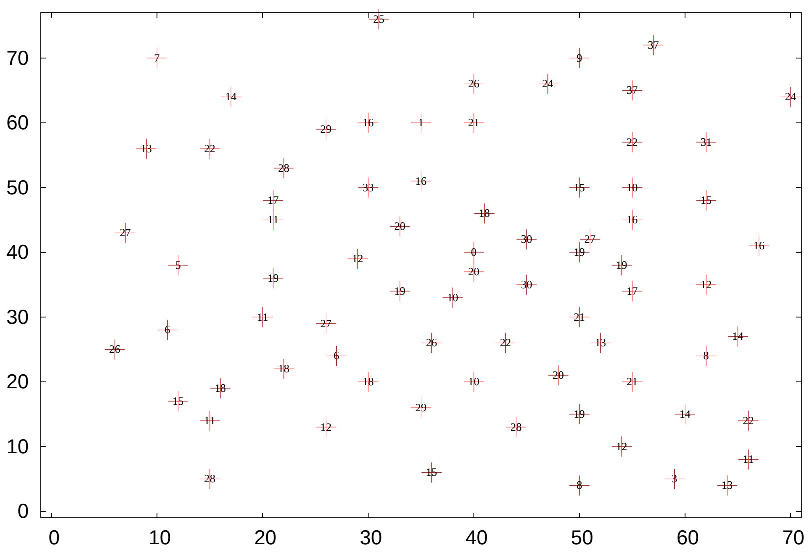

E-n51-k5
0
10
20
30
40
50
60
70


E-n76-k10
0
10
20
30
40
50
60
70

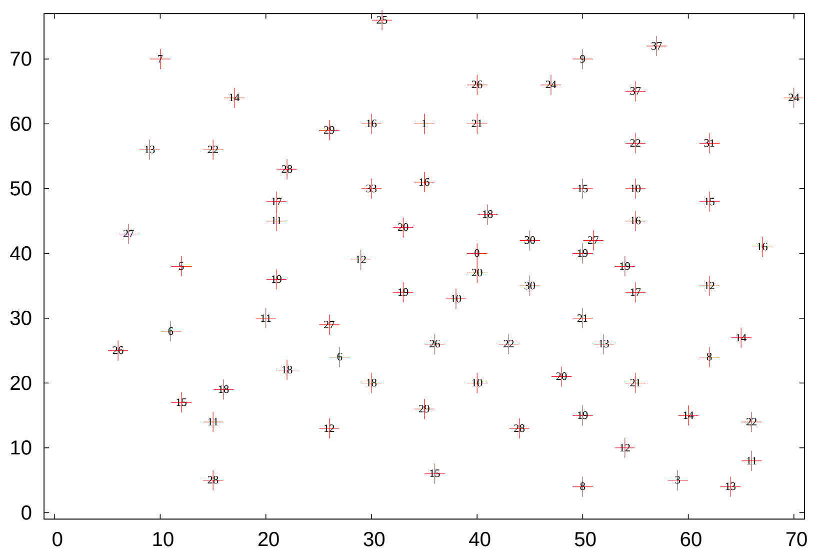
E-n76-k14



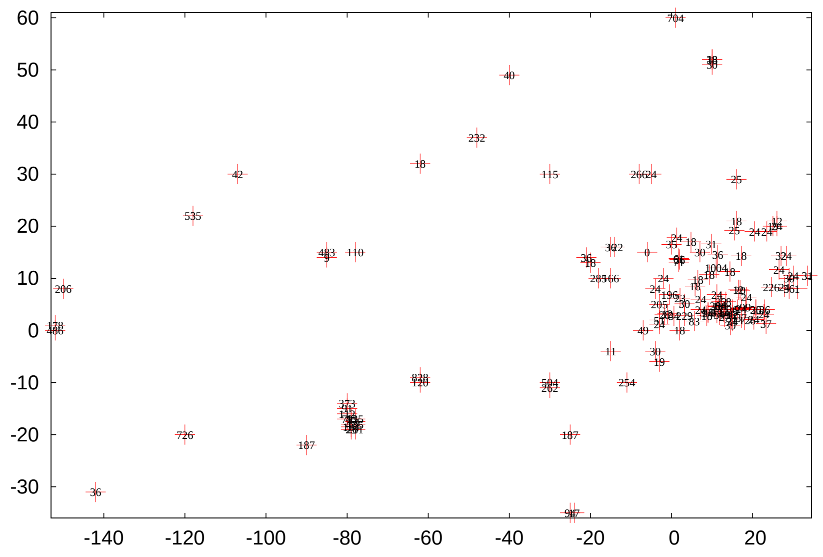
E-n76-k7



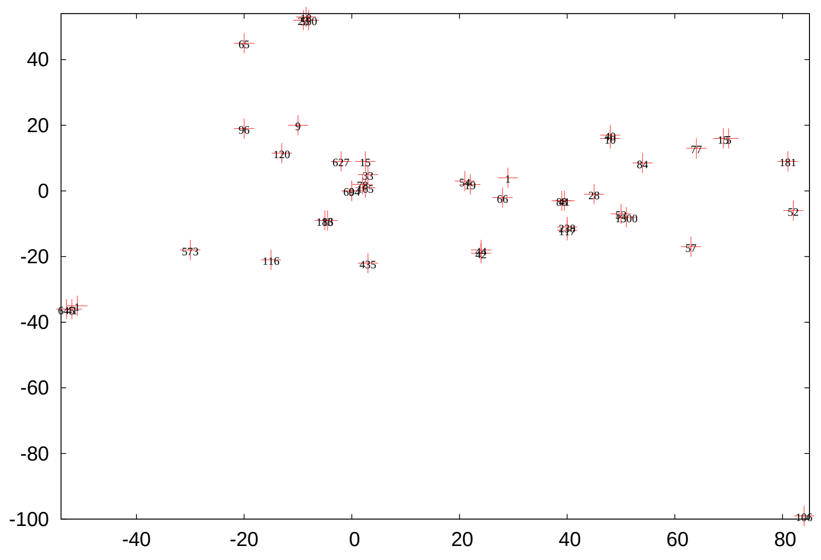
E-n76-k8



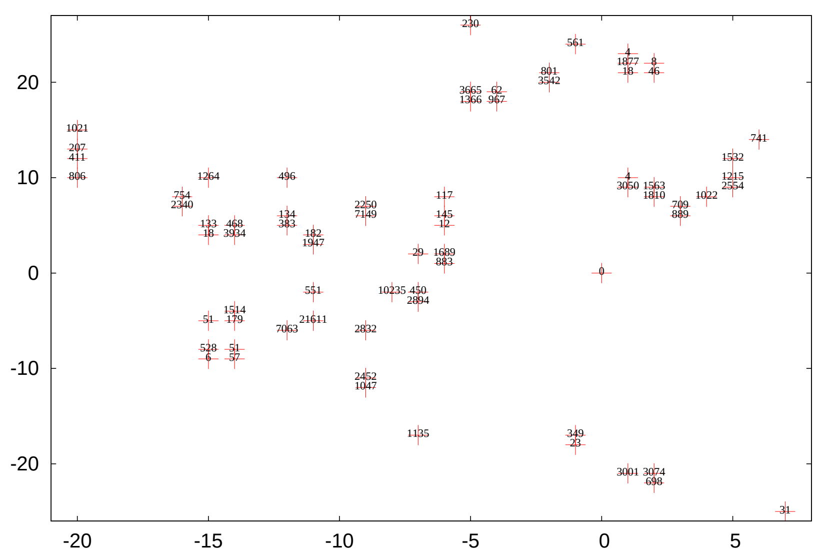
F-n135-k7



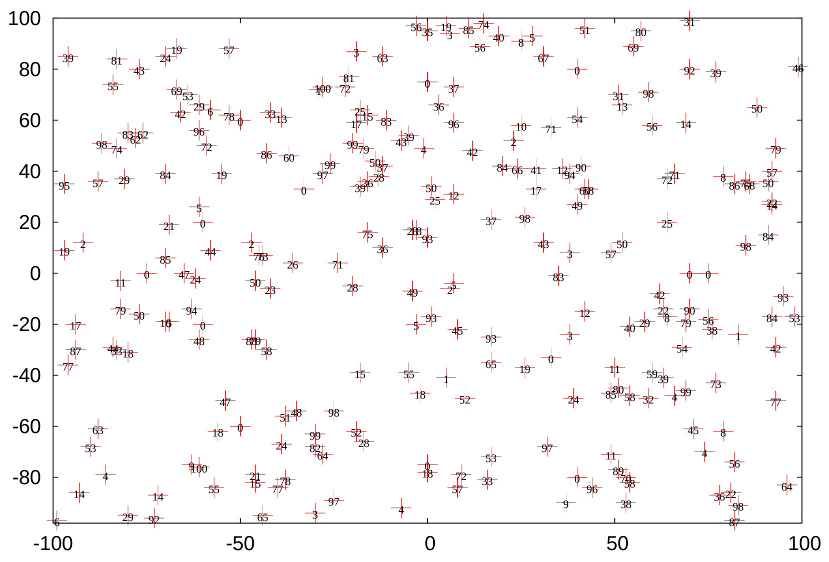
F-n45-k4



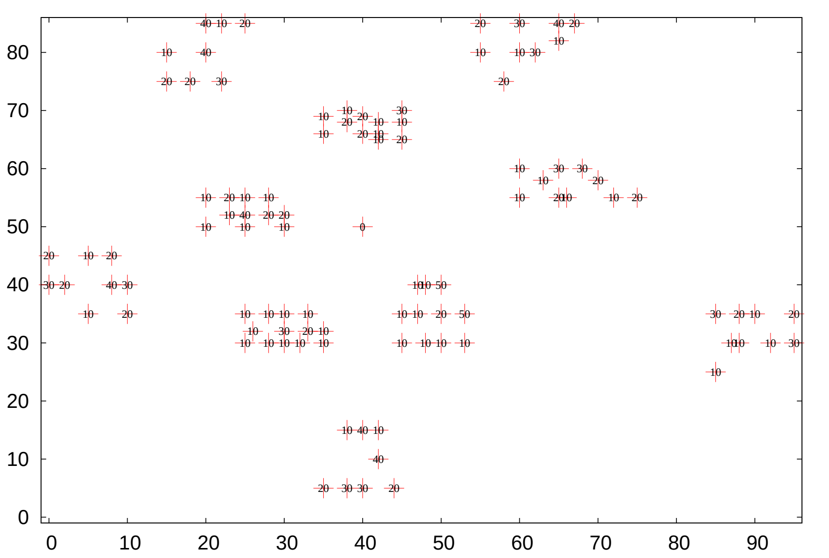
F-n72-k4



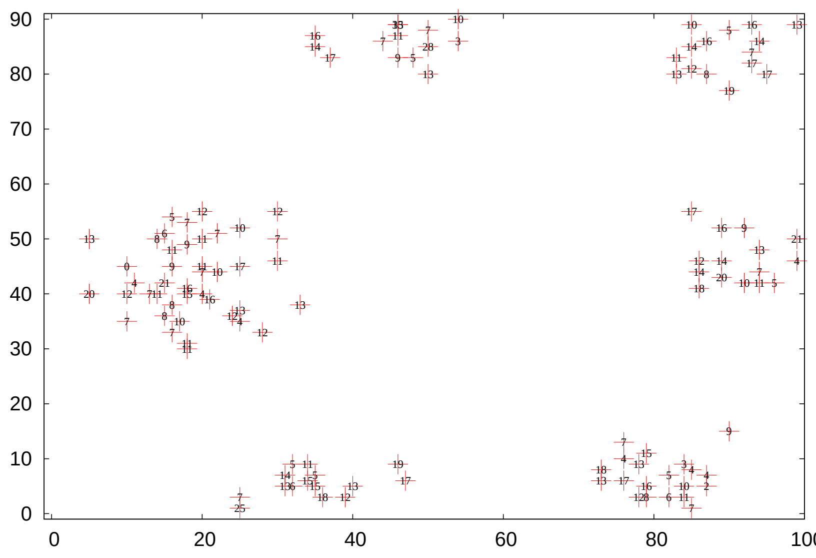
G-n262-k25



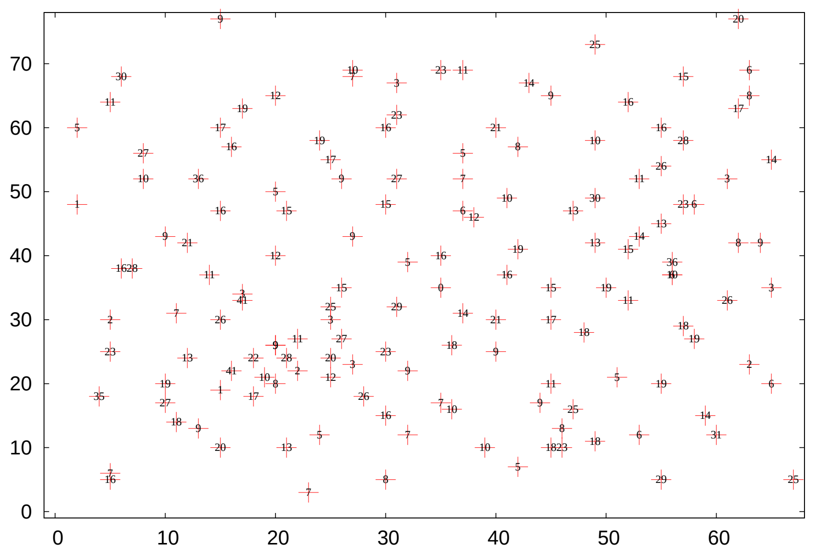
M-n101-k10



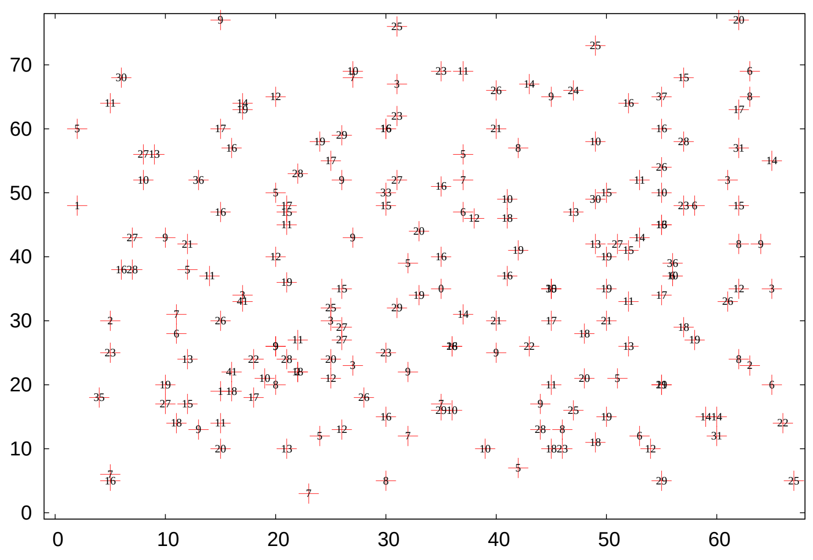
M-n121-k7



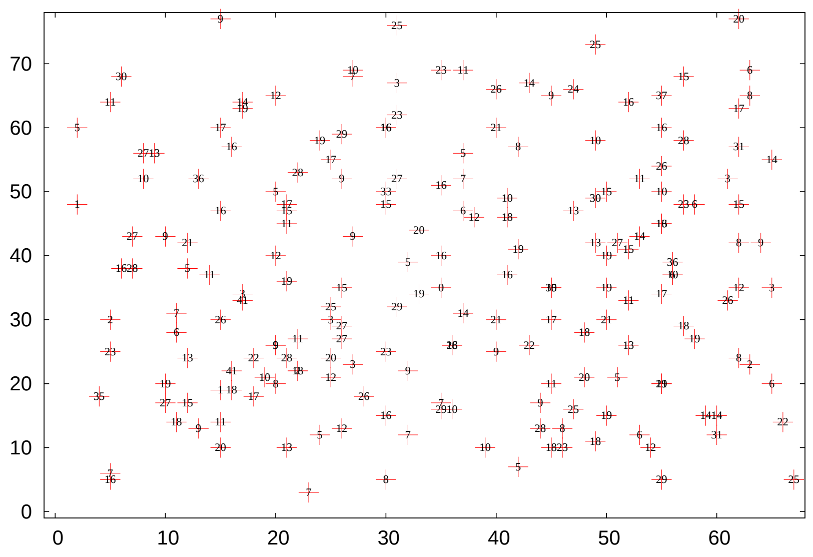
M-n151-k12



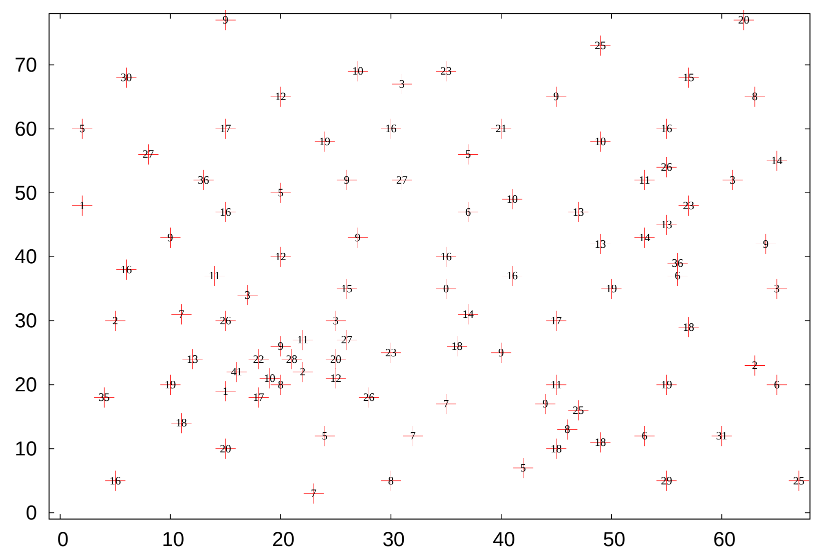
M-n200-k16



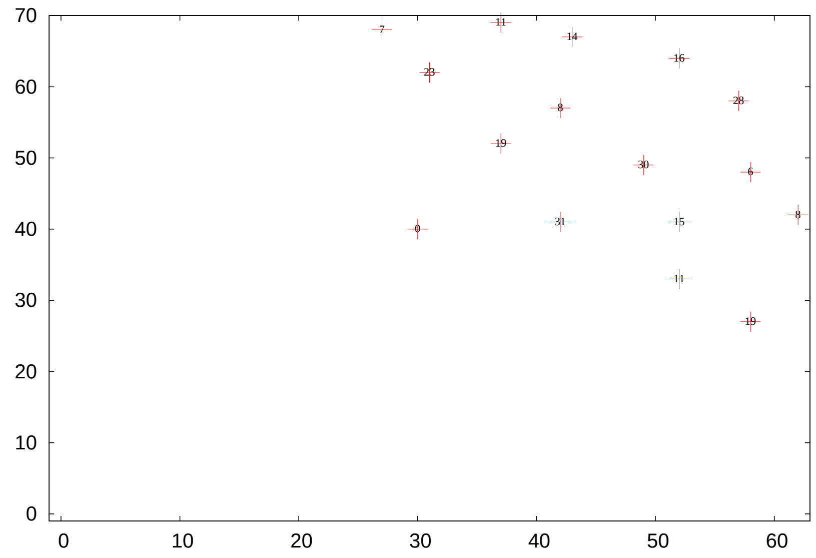
M-n200-k17

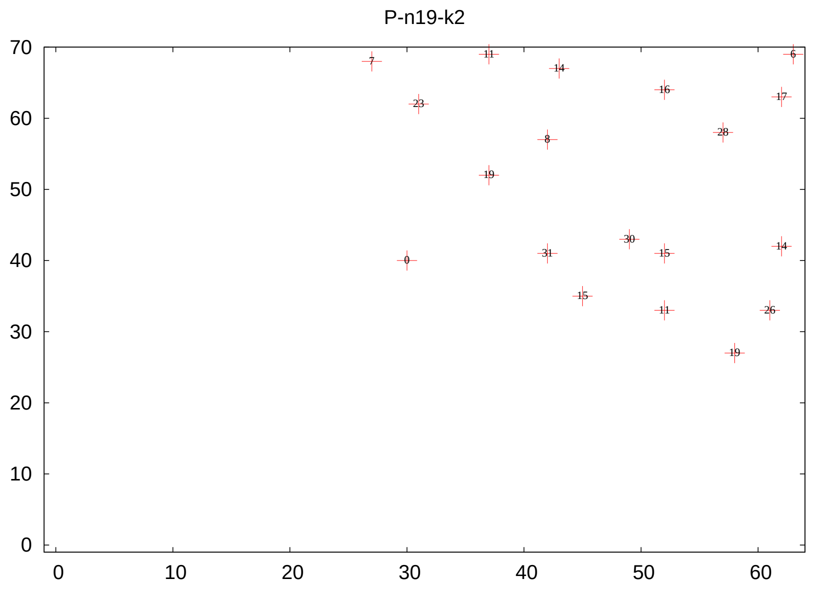


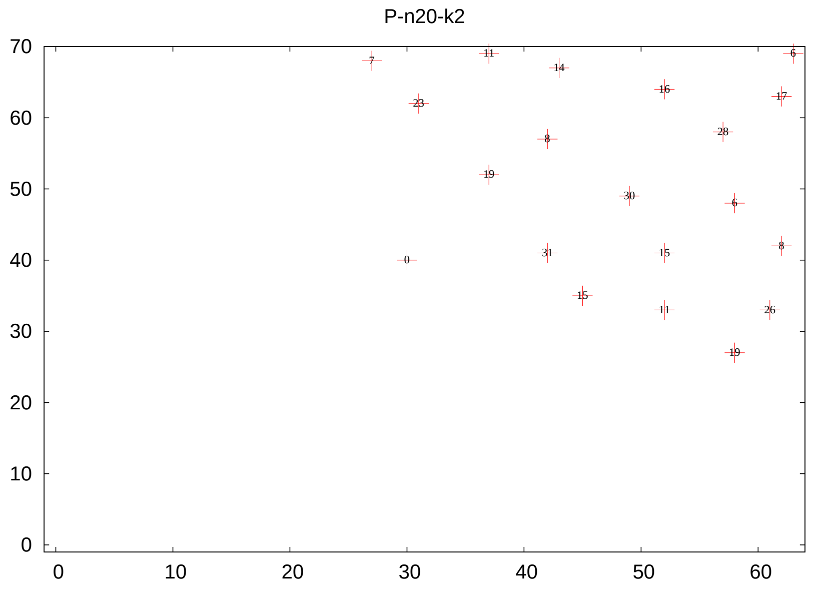
P-n101-k4



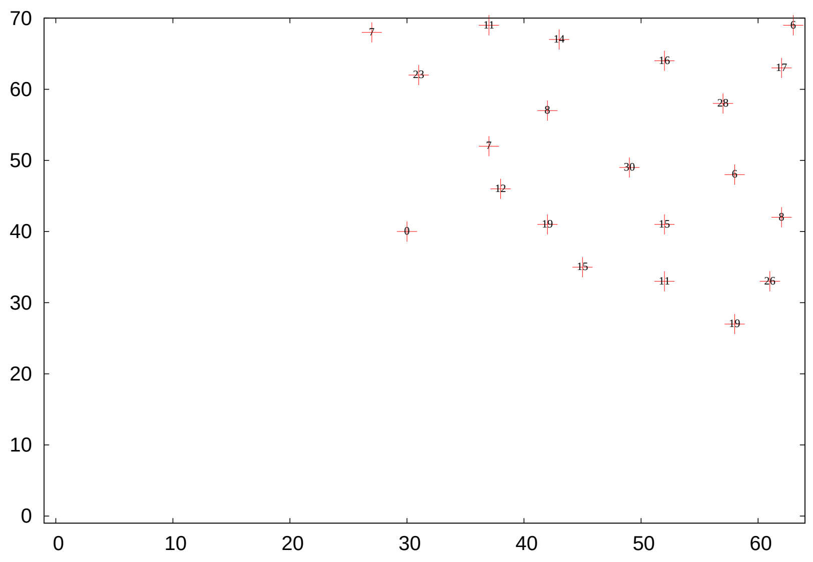
P-n16-k8



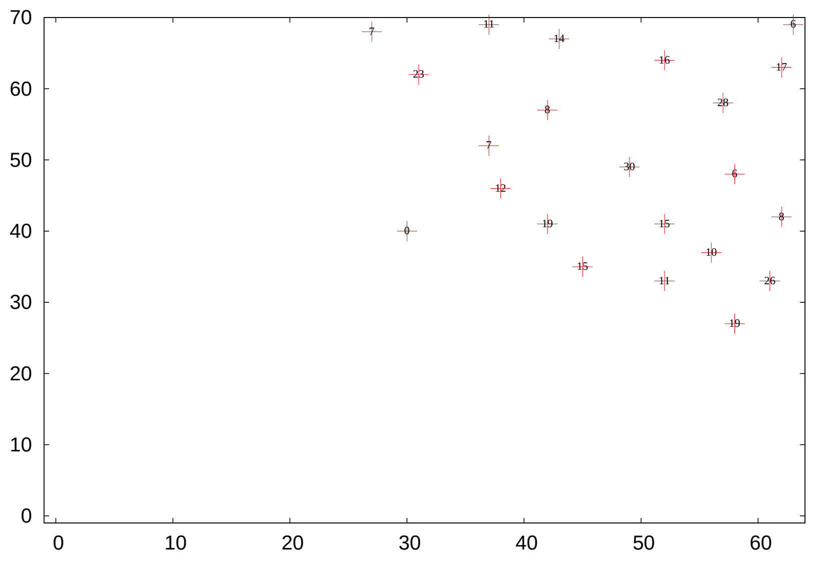

P-n19-k2
7
11
14
6
16
23
17
28
8
19
30
31
15
14
0
15
11
26
19
0
10
20
30
40
50
60
70


P-n20-k2
7
11
14
6
16
23
17
28
8
19
30
6
31
15
8
0
15
11
26
19
0
10
20
30
40
50
60
70

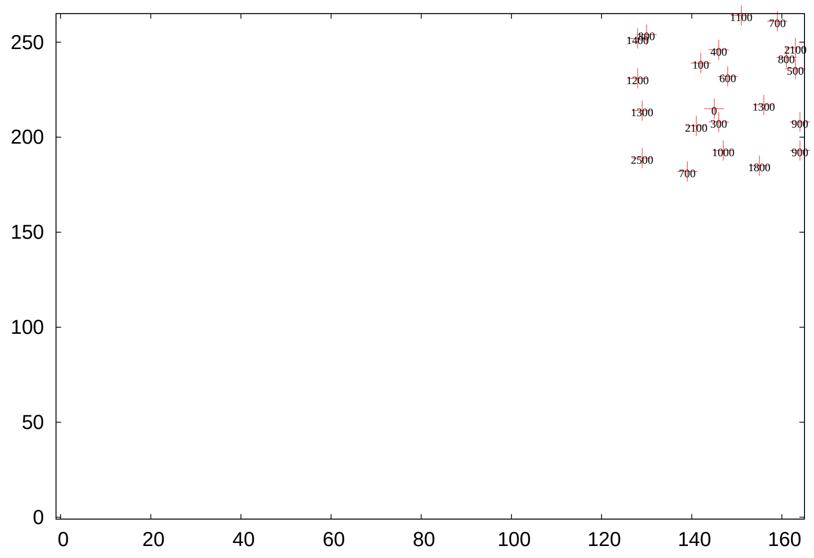
P-n21-k2



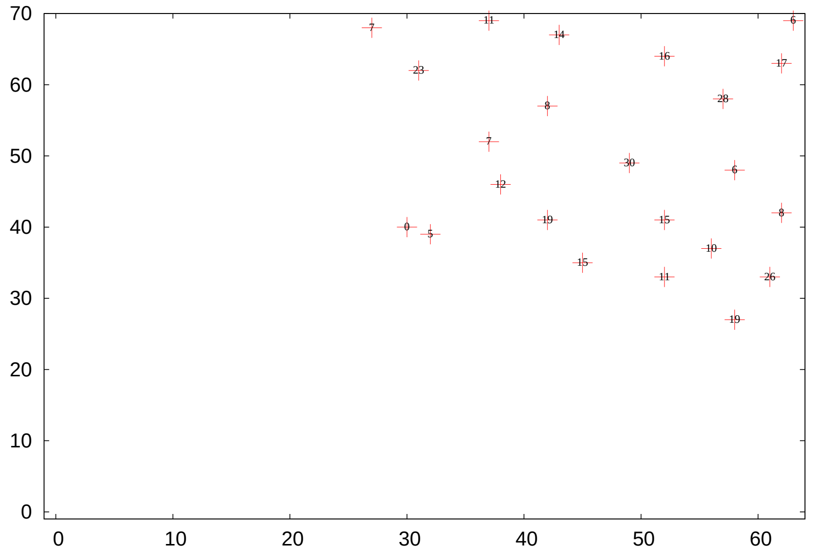
P-n22-k2

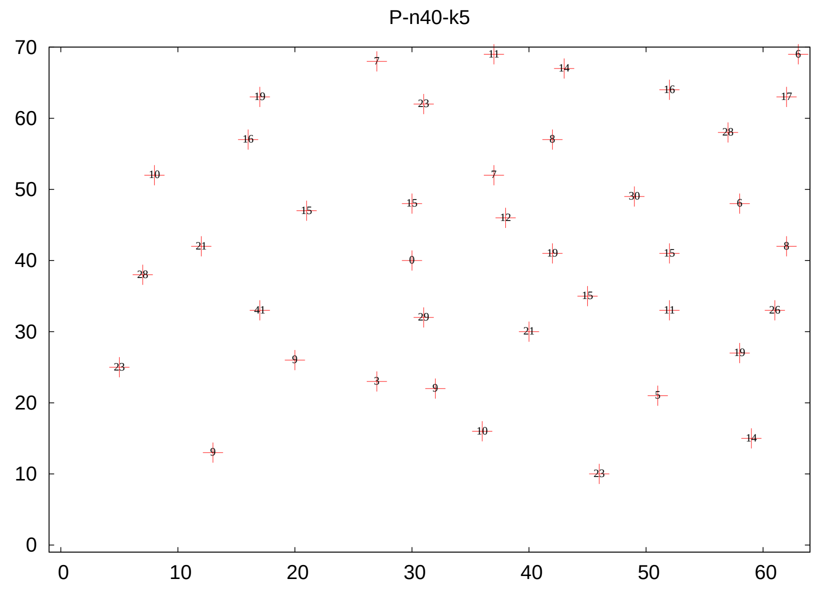


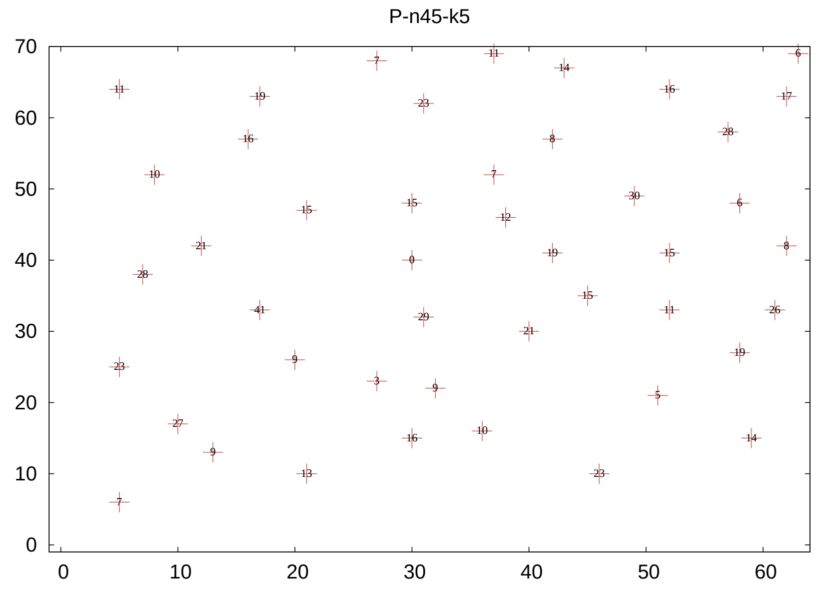
P-n22-k8



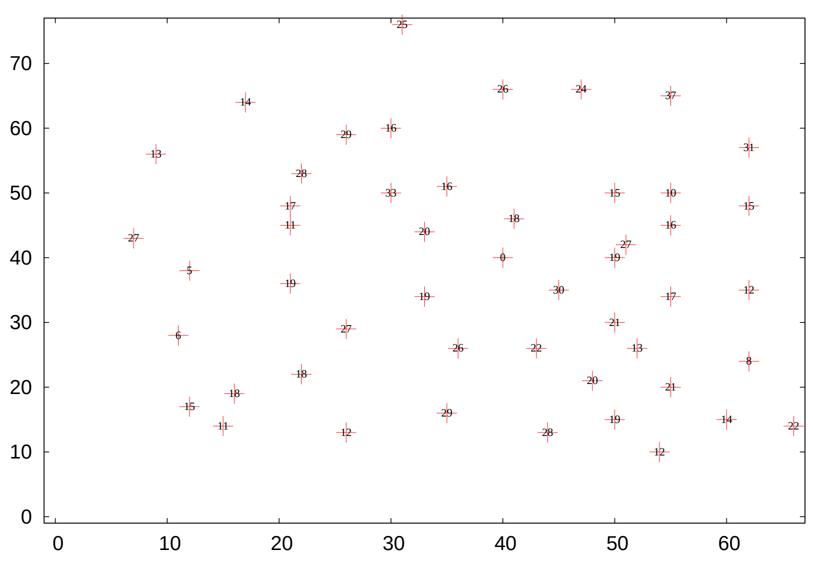
P-n23-k8



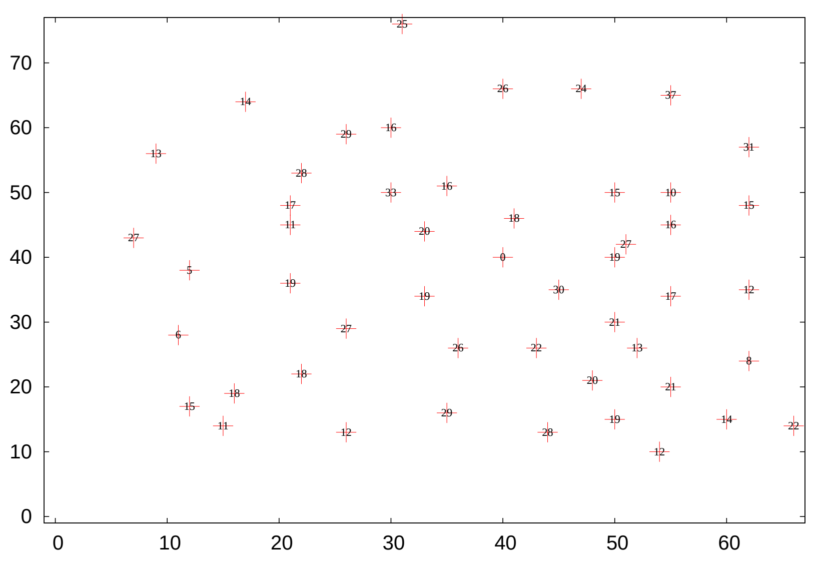

P-n40-k5


P-n45-k5

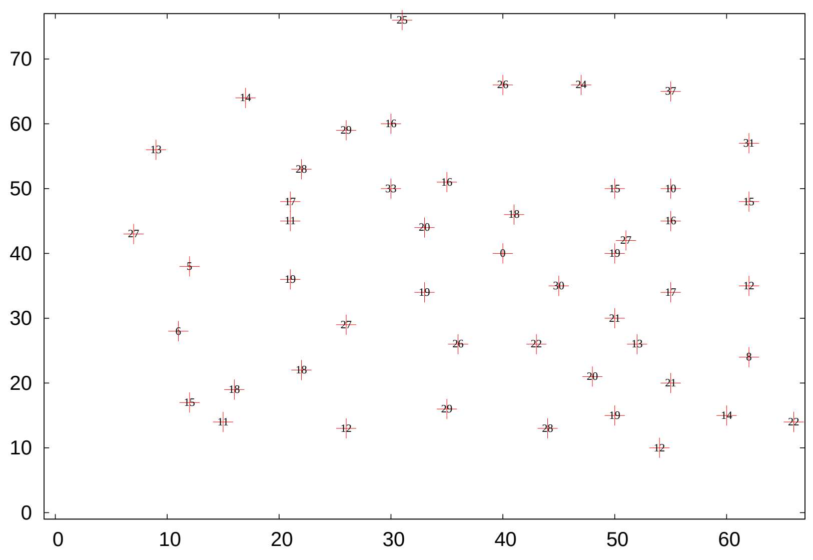
P-n50-k10



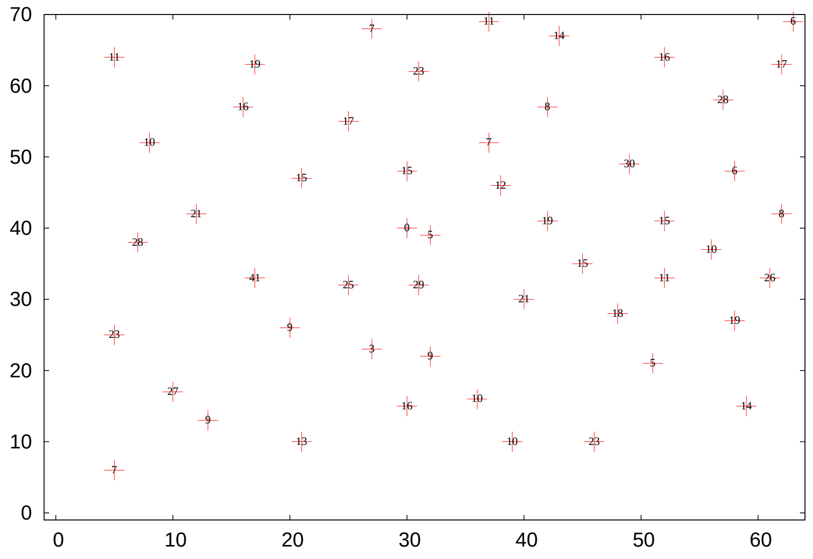
P-n50-k7



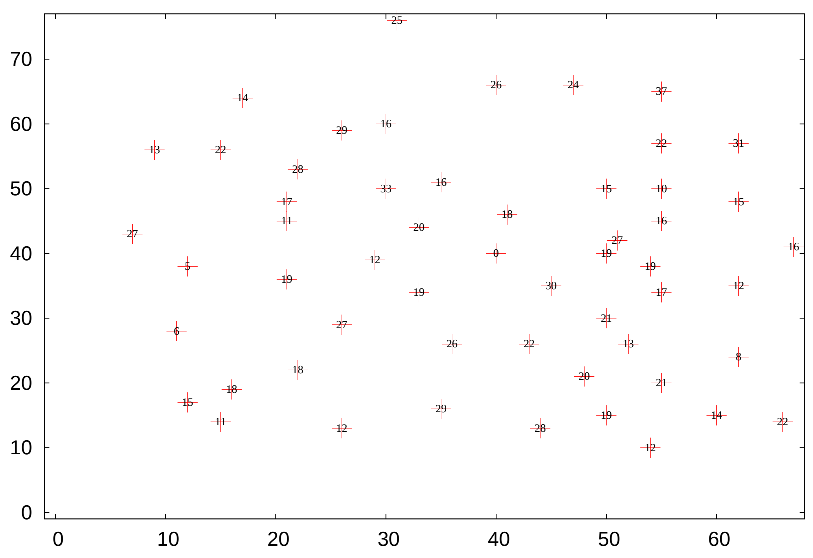
P-n50-k8



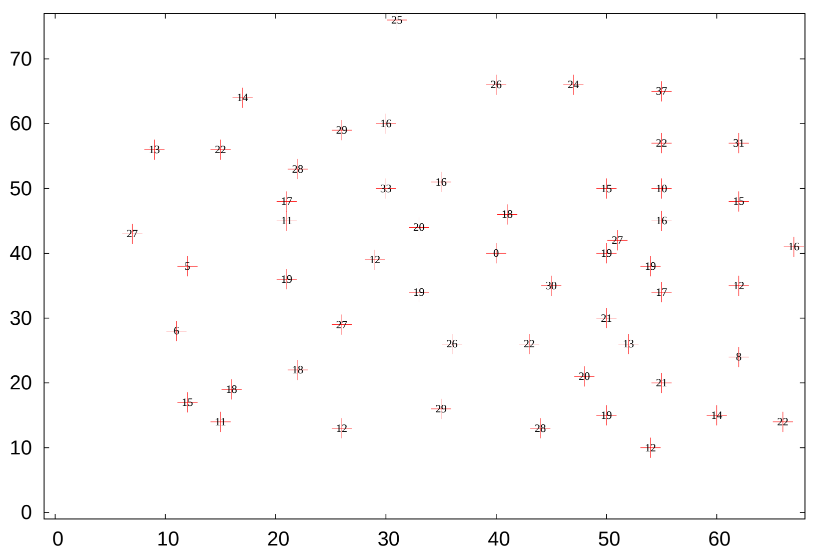
P-n51-k10



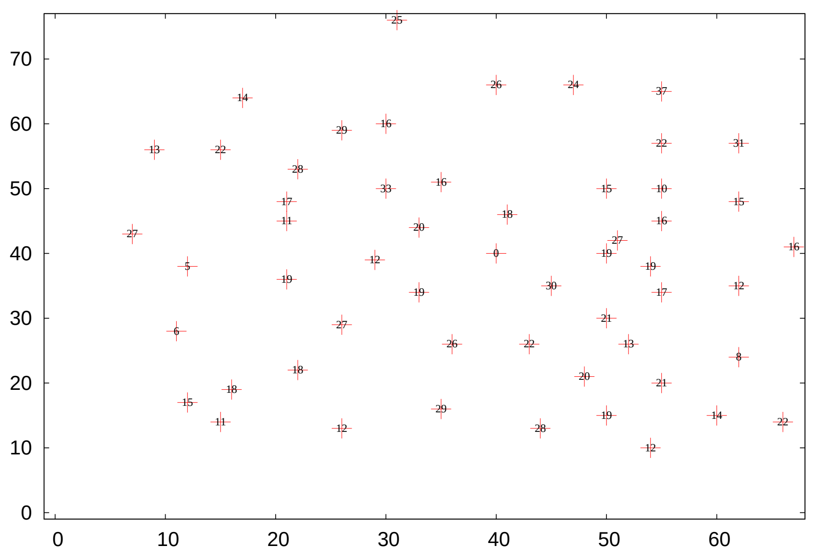
P-n55-k10



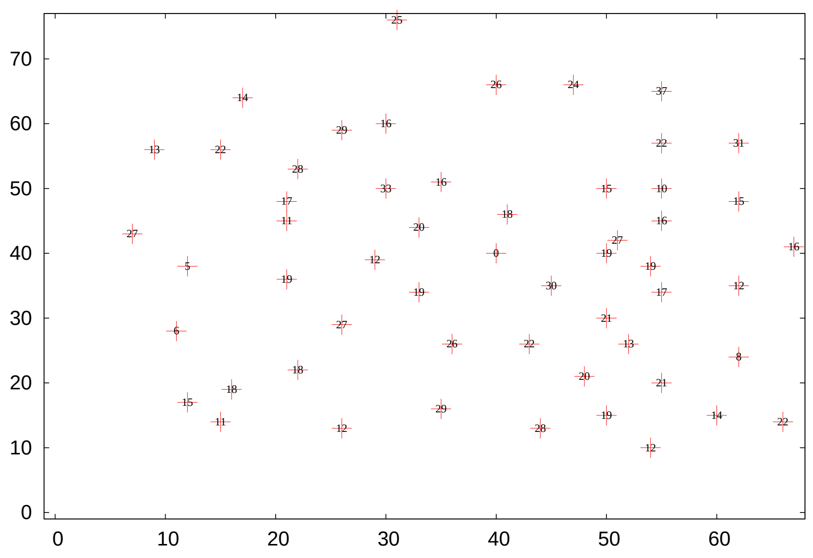
P-n55-k15



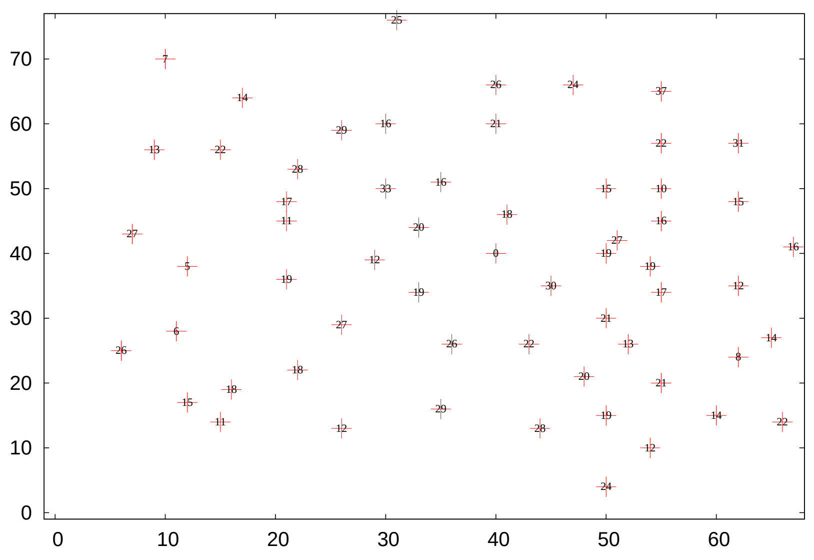
P-n55-k7



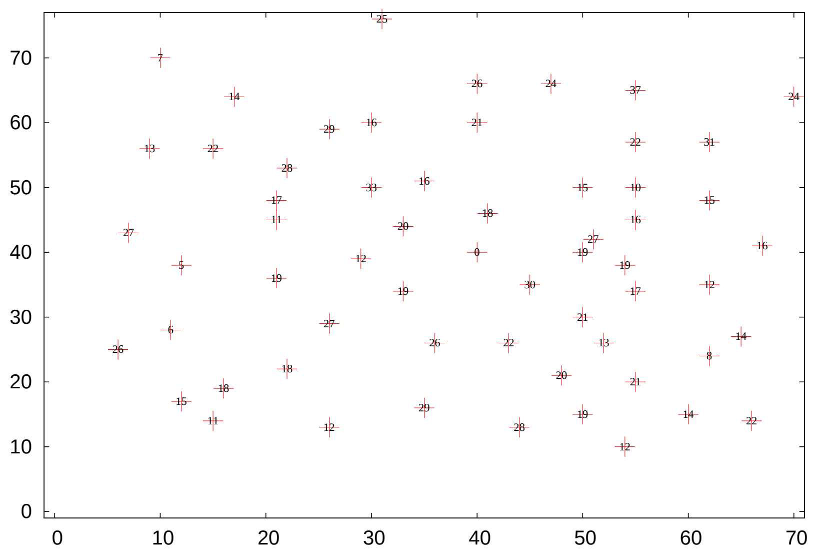
P-n55-k8



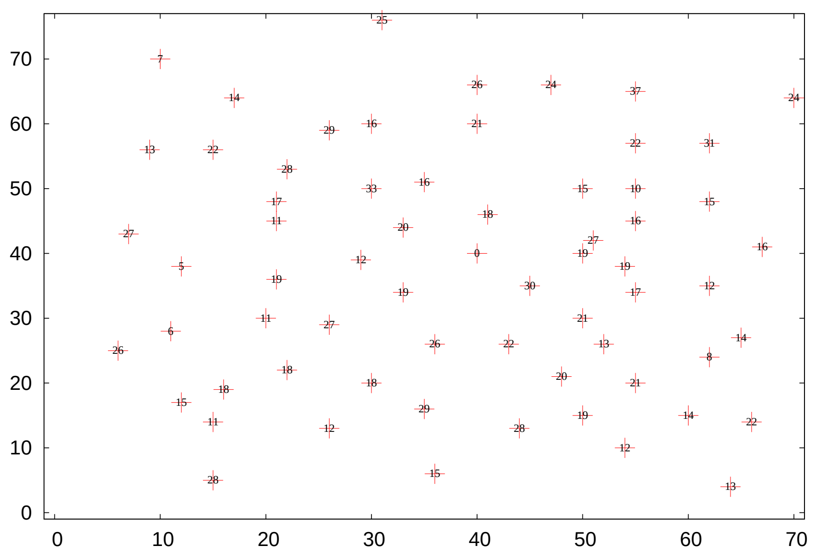
P-n60-k10



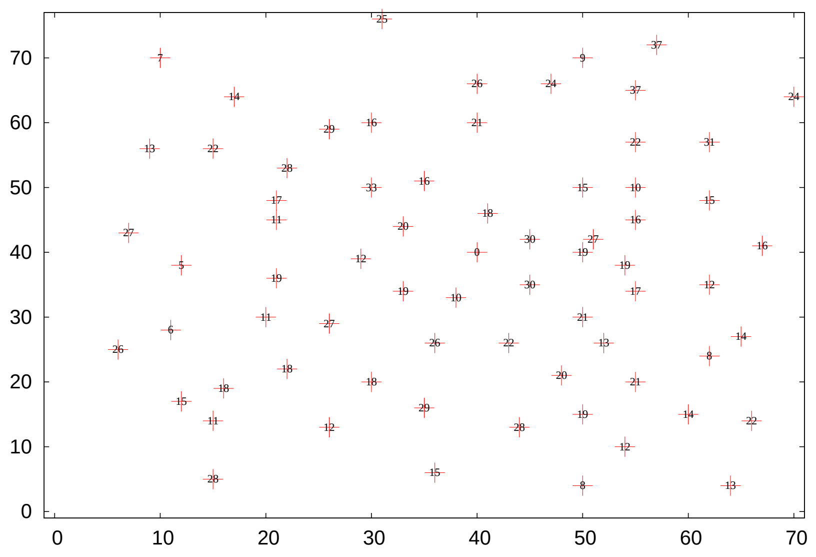
P-n60-k15



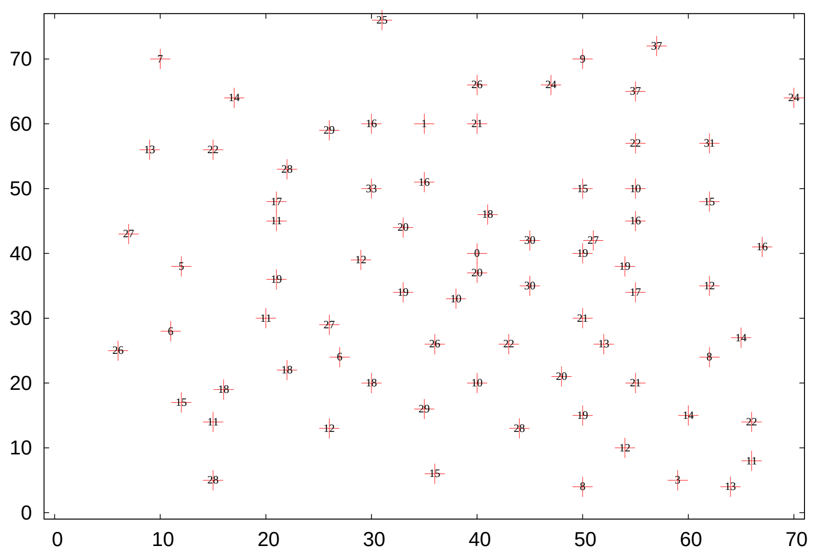
P-n65-k10



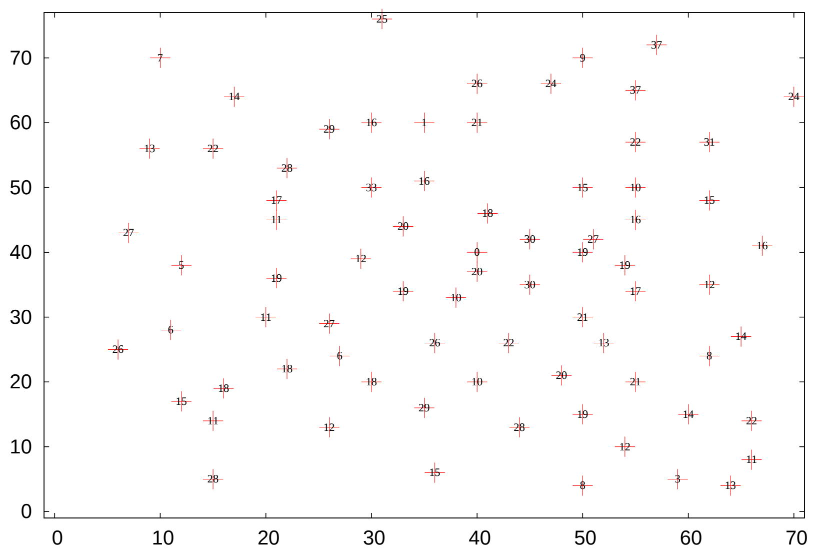
P-n70-k10



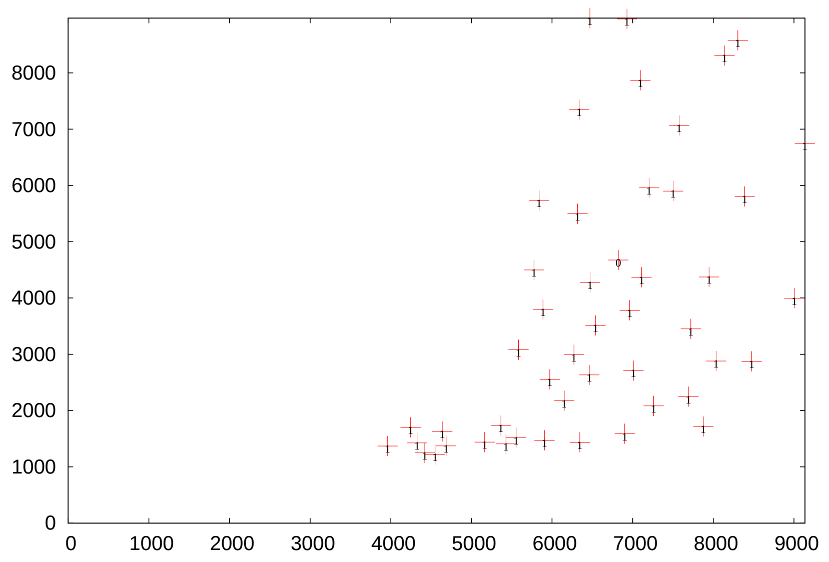
P-n76-k4

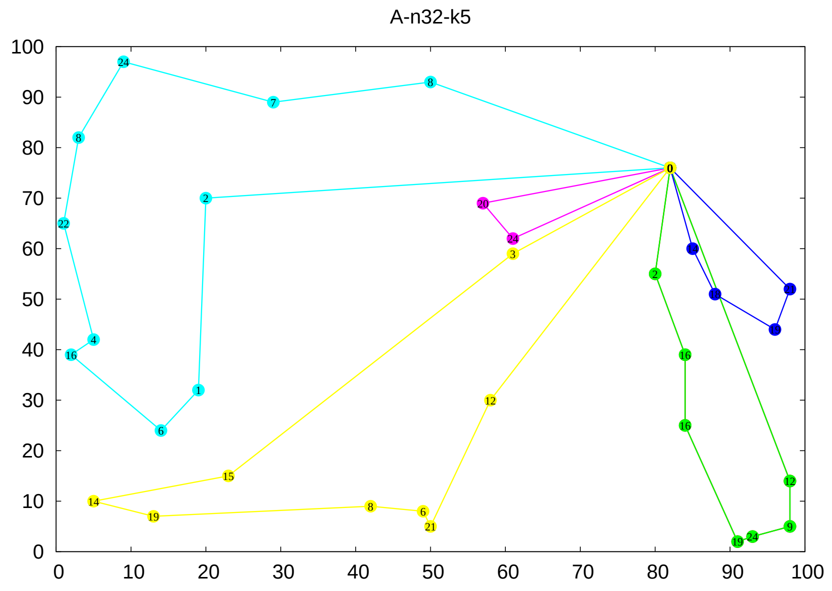


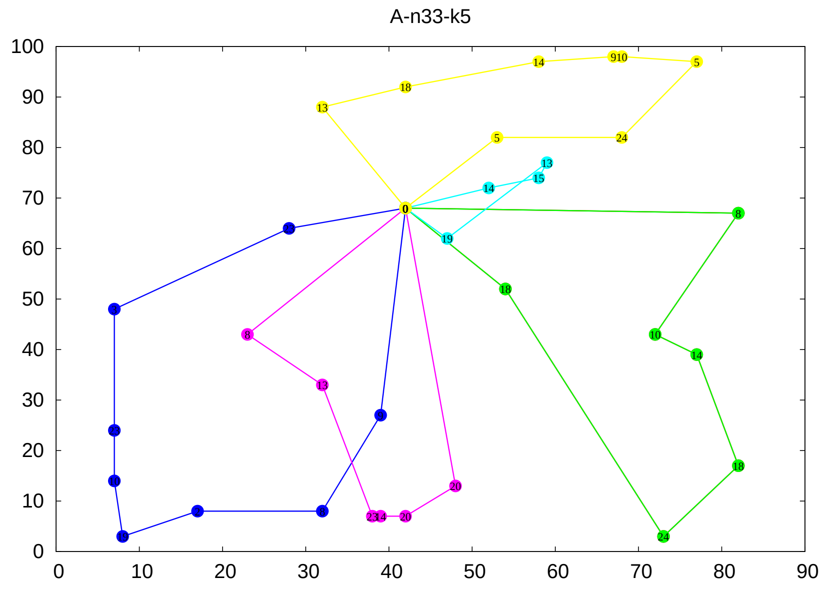
P-n76-k5



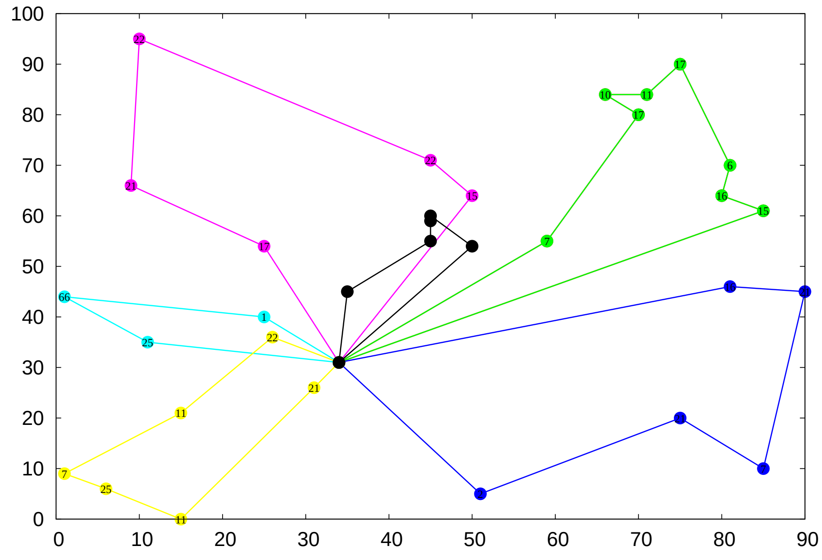
att-n48-k4


0
1000
2000
3000
4000
5000
6000
7000
8000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
0

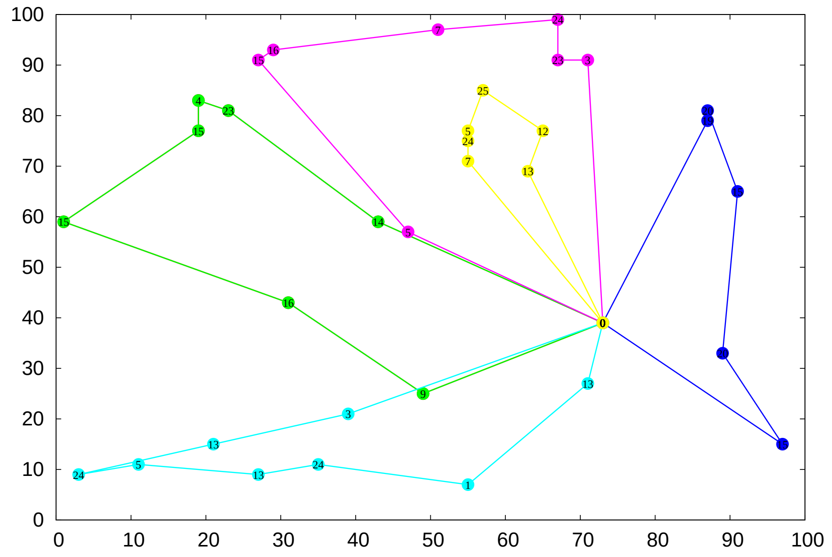

A-n32-k5

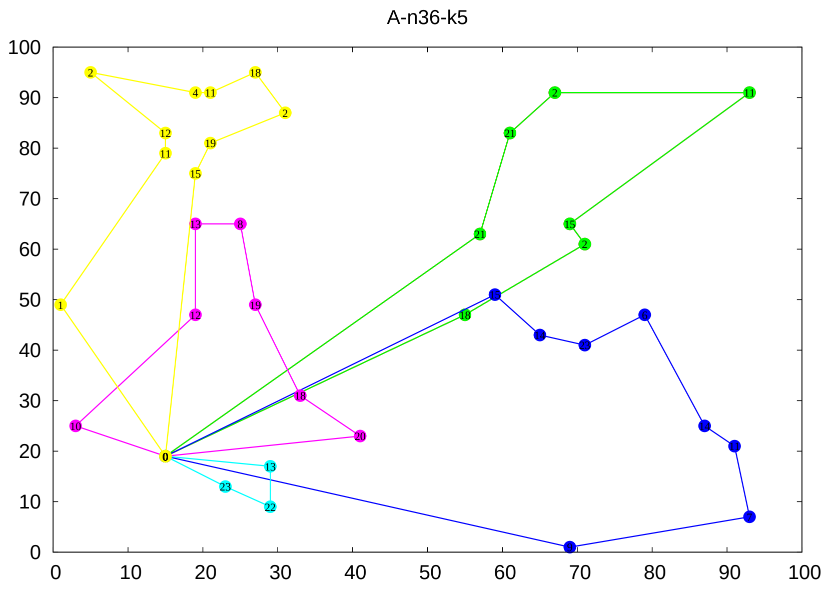

A-n33-k5

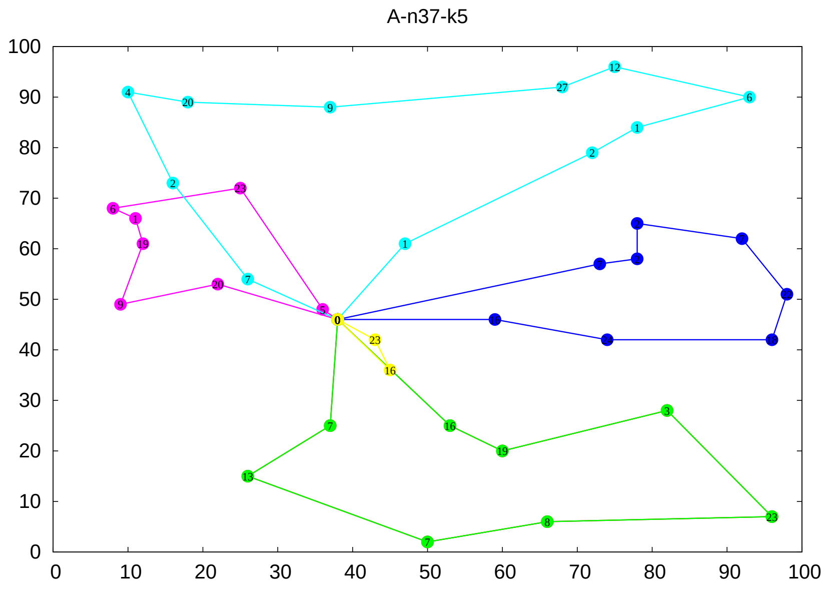
A-n33-k6



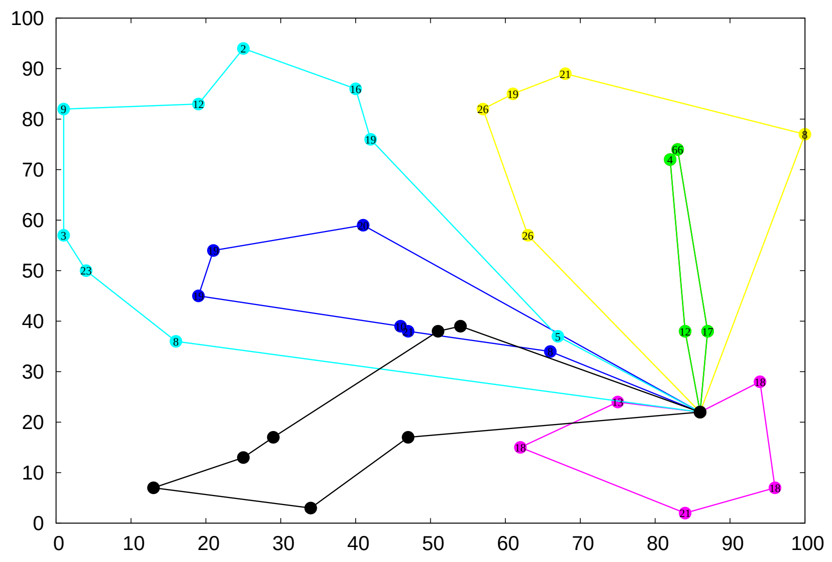
A-n34-k5



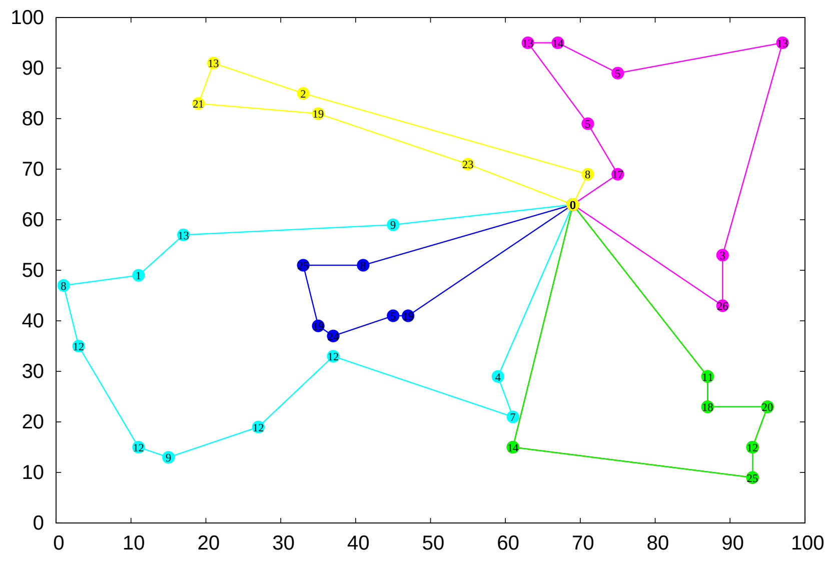

A-n36-k5
0
10
20
30
40
50
60
70
80
90
100


A-n37-k5
0
10
20
30
40
50
60
70
80
90
100

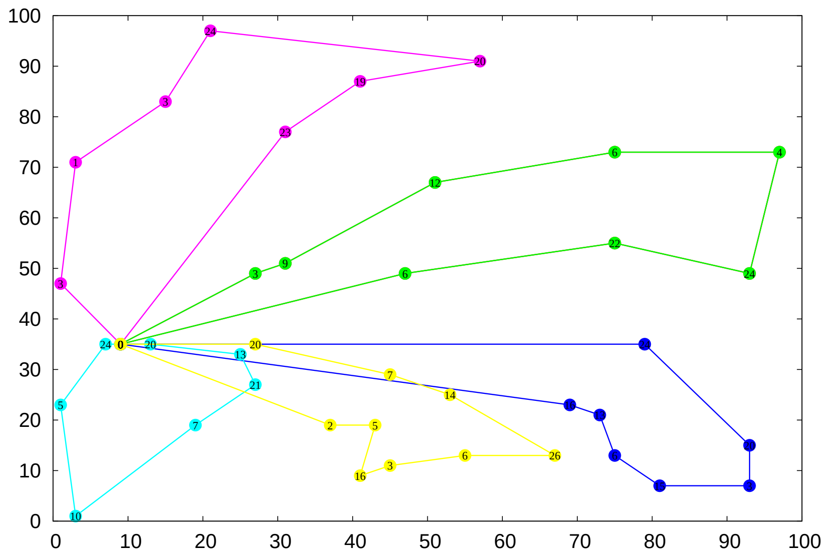
A-n37-k6



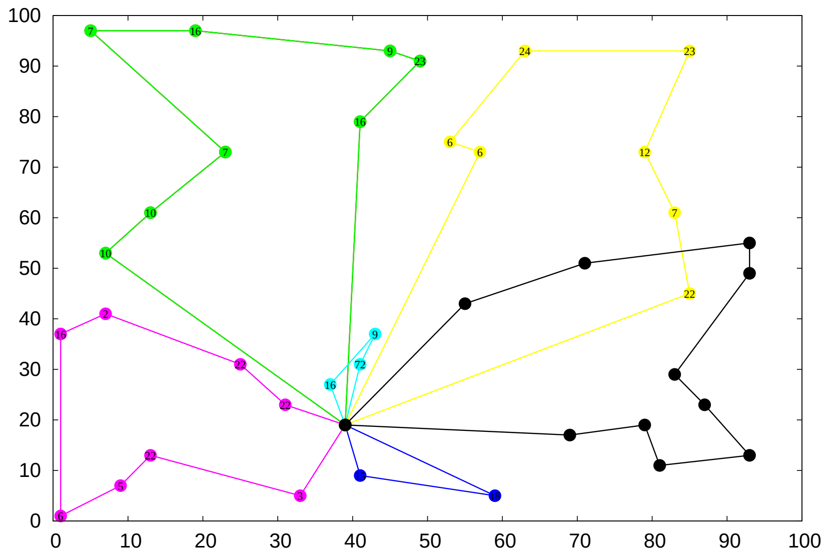
A-n38-k5

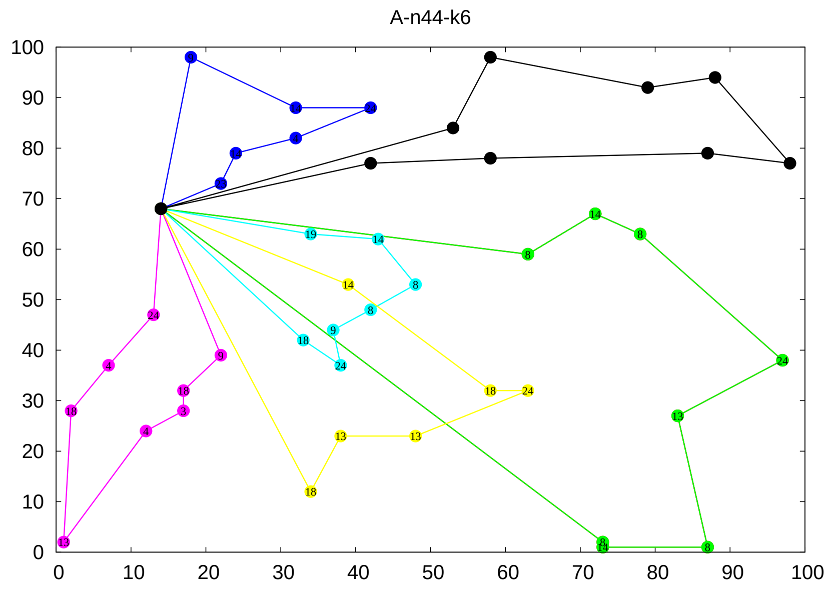


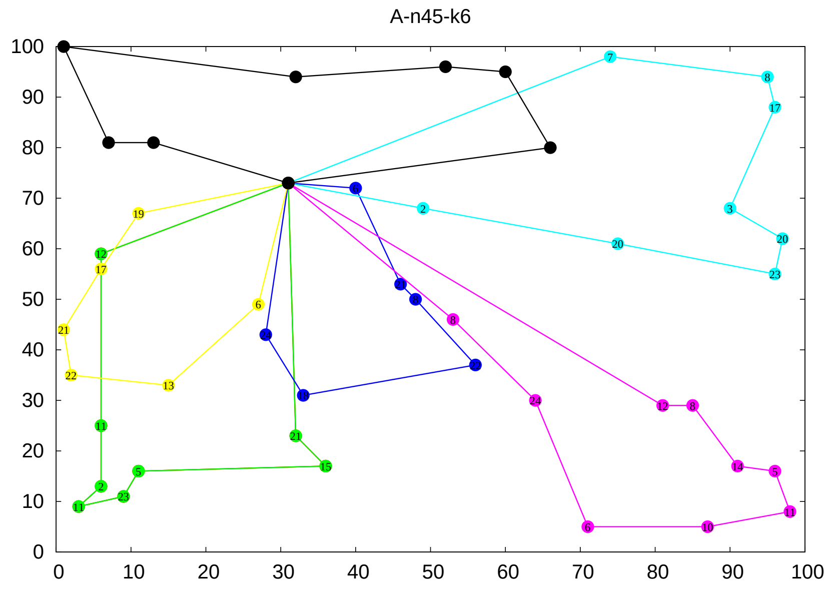
A-n39-k5



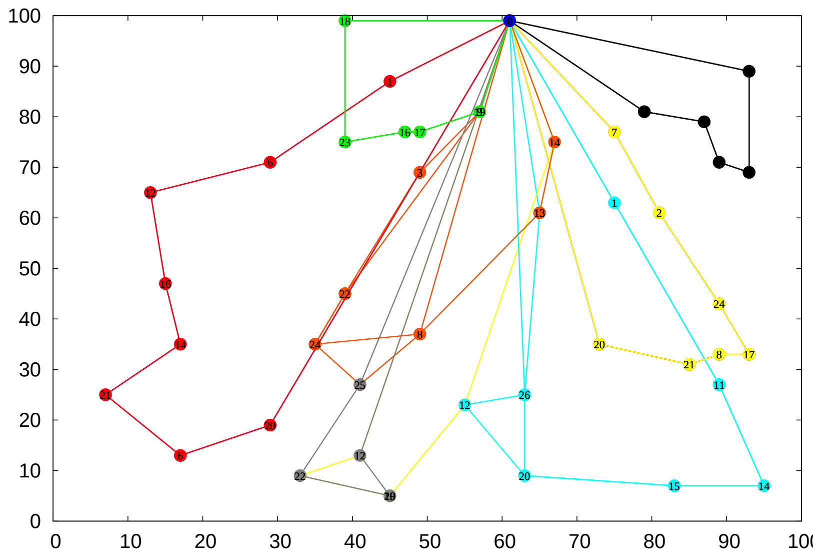
A-n39-k6



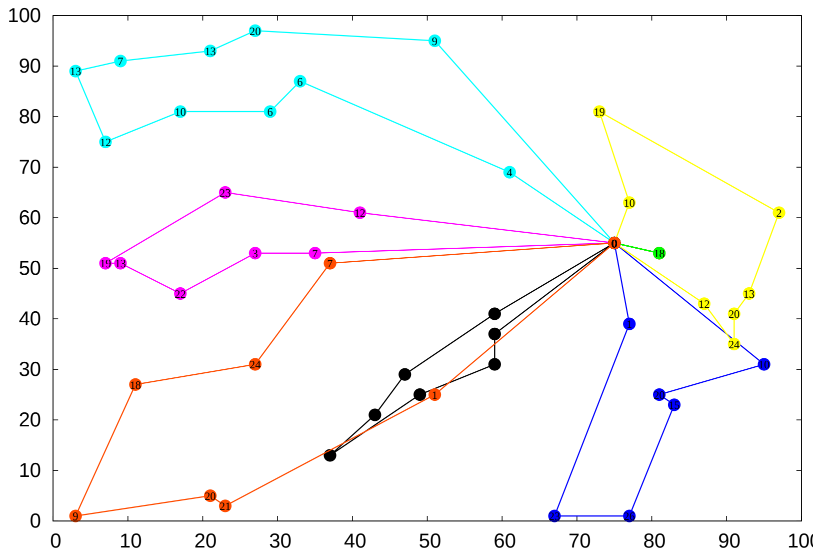

A-n44-k6


A-n45-k6

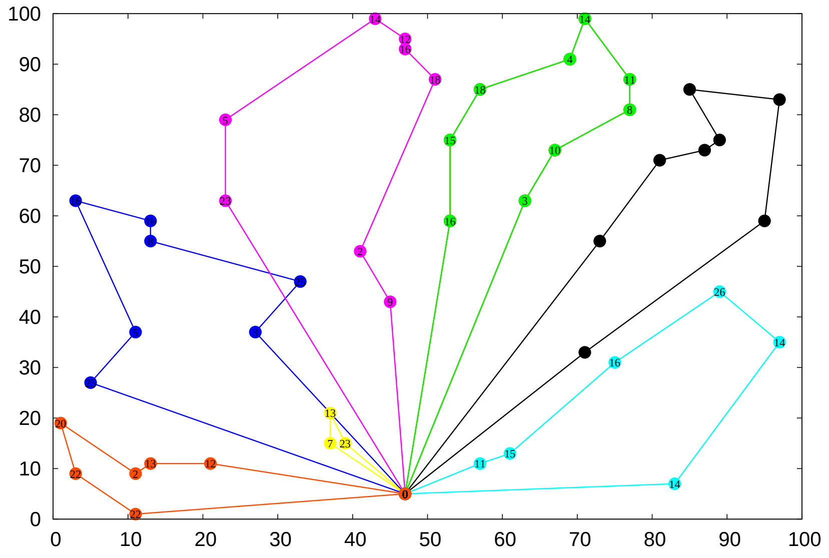
A-n45-k7



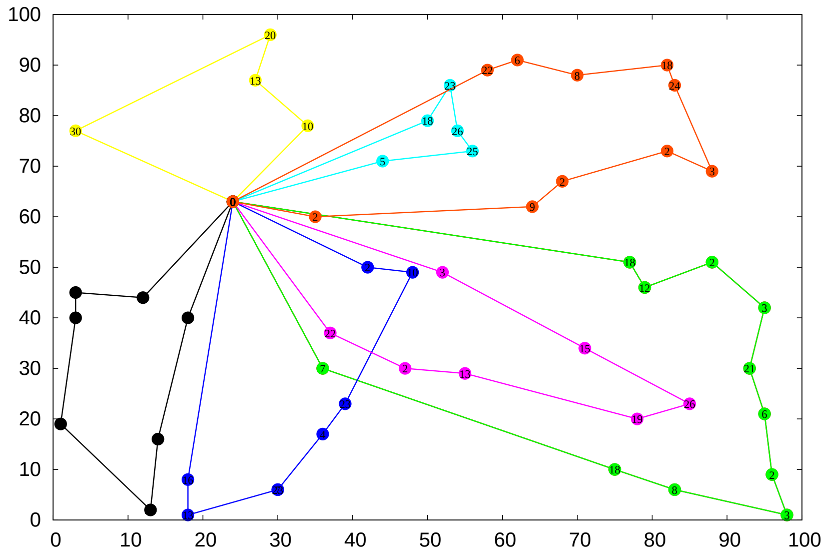
A-n46-k7

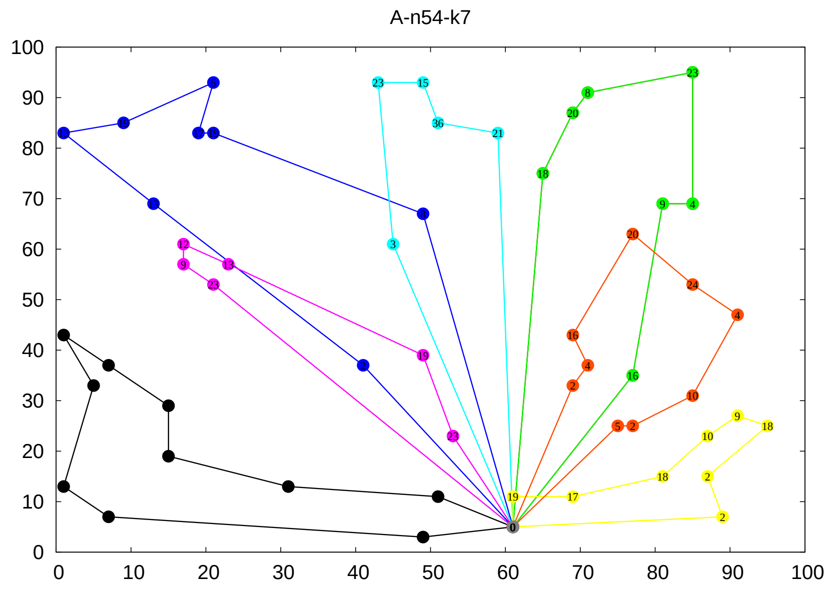


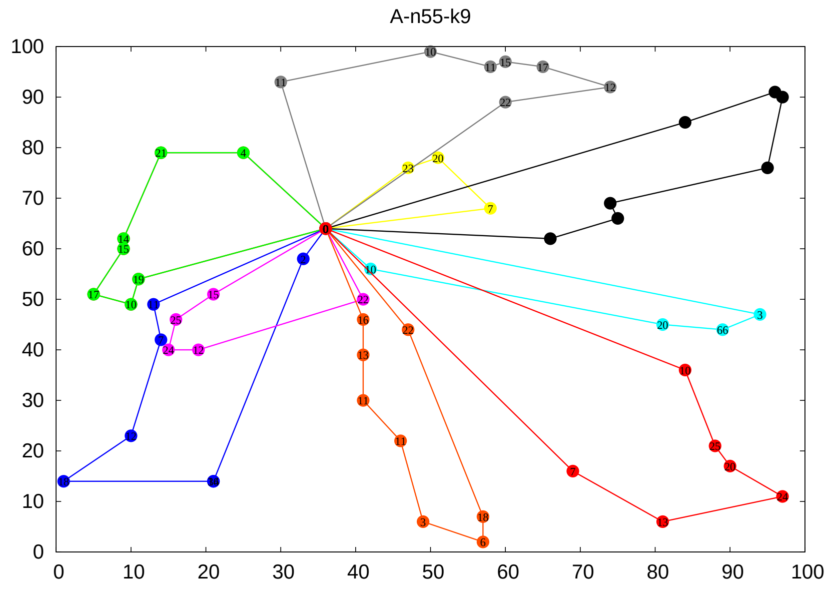
A-n48-k7



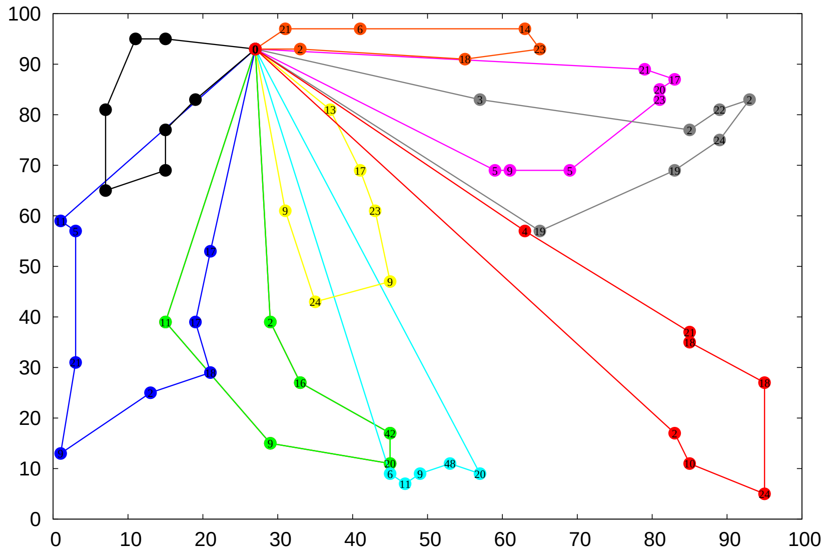
A-n53-k7



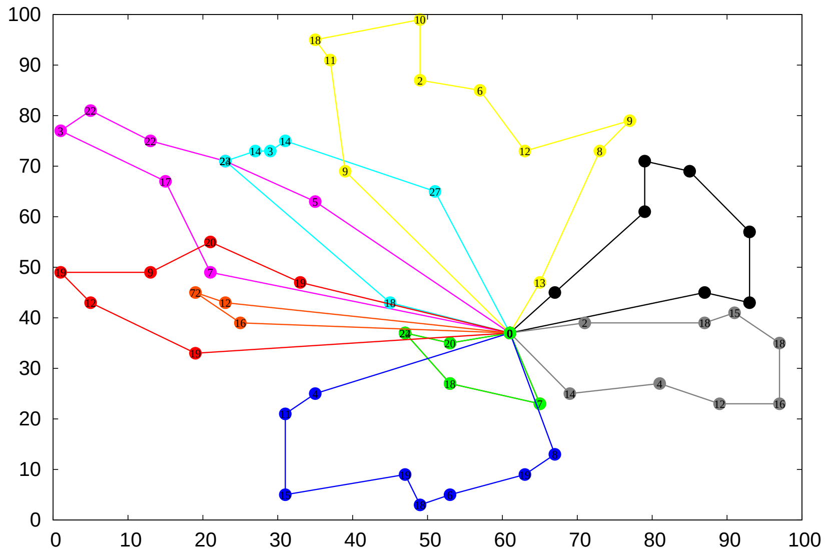

A-n54-k7


A-n55-k9

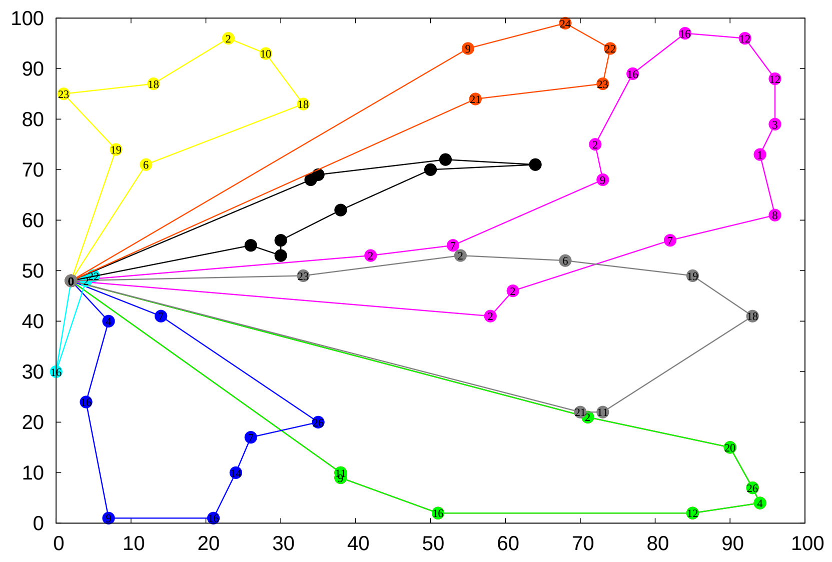
A-n60-k9



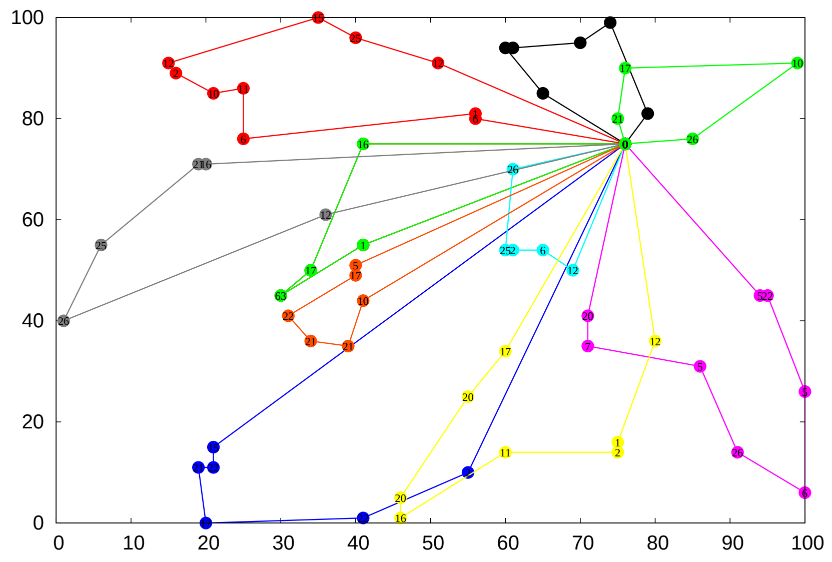
A-n61-k9

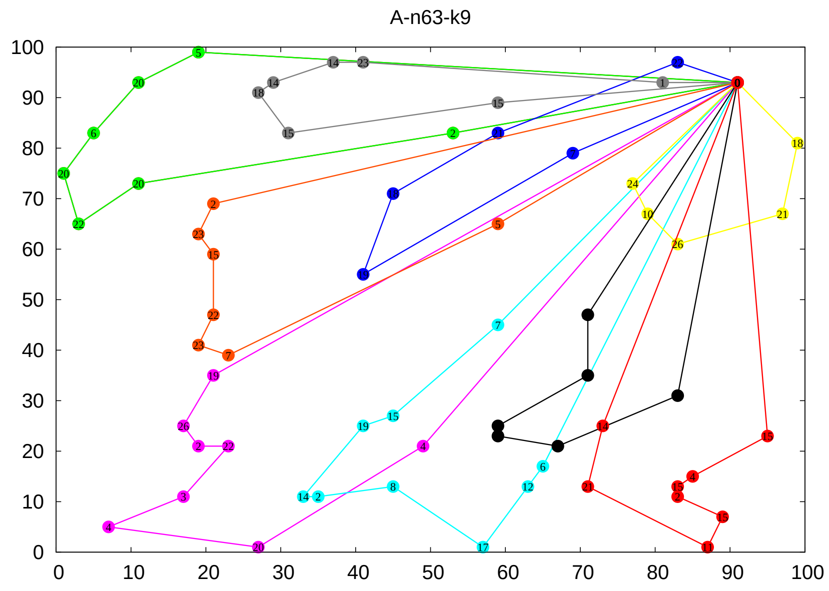


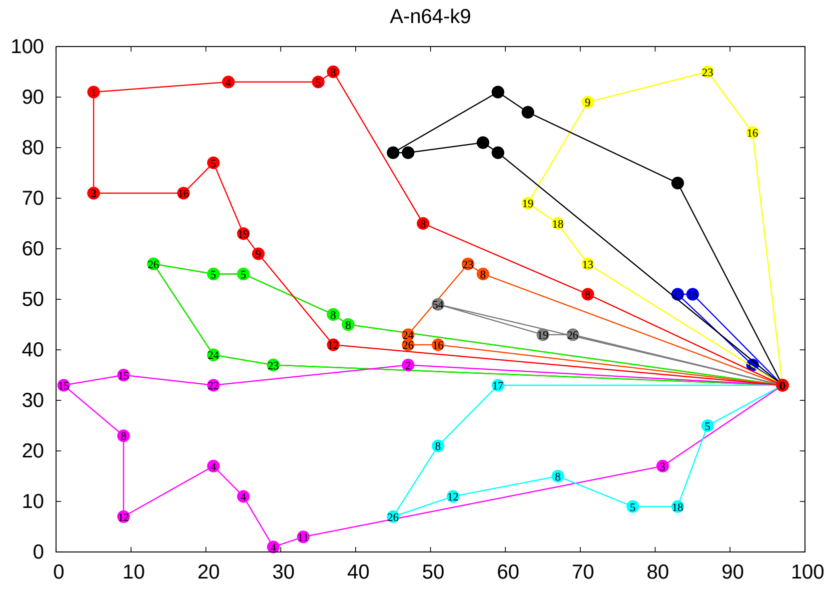
A-n62-k8



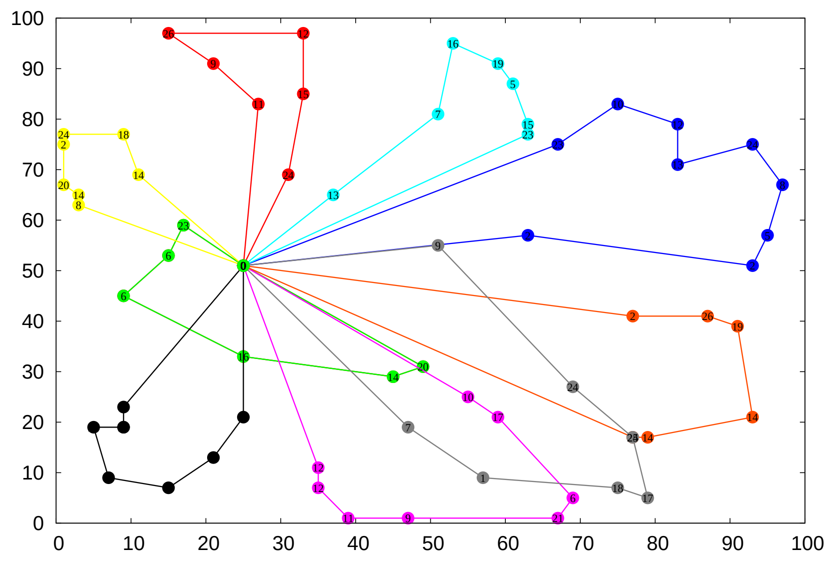
A-n63-k10



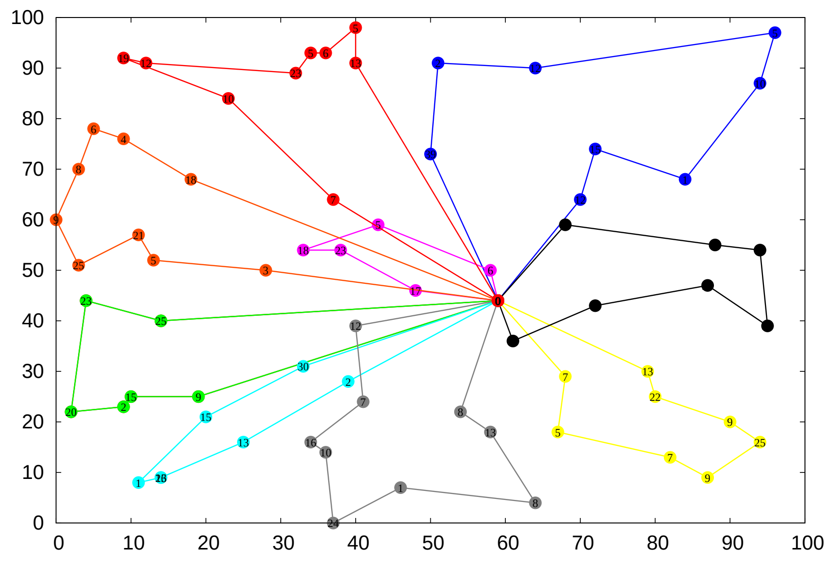

A-n63-k9
0
10
20
30
40
50
60
70
80
90
100


A-n64-k9
0
10
20
30
40
50
60
70
80
90
100

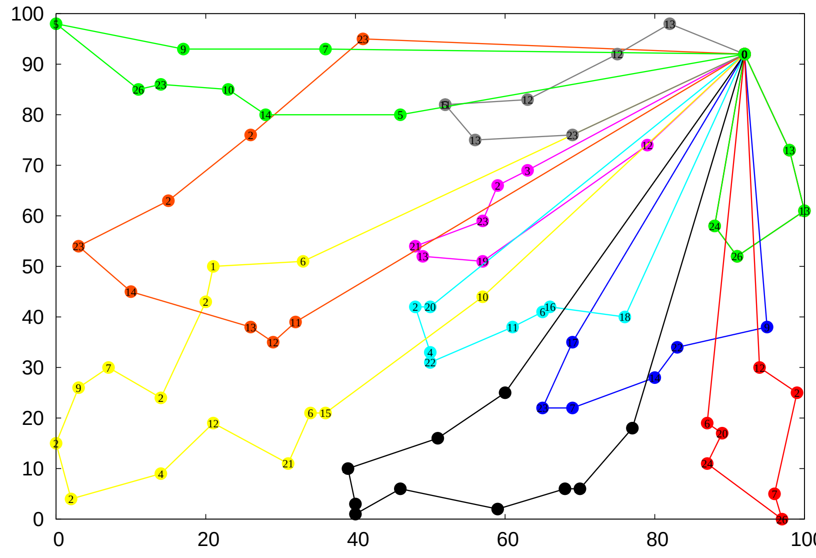
A-n65-k9



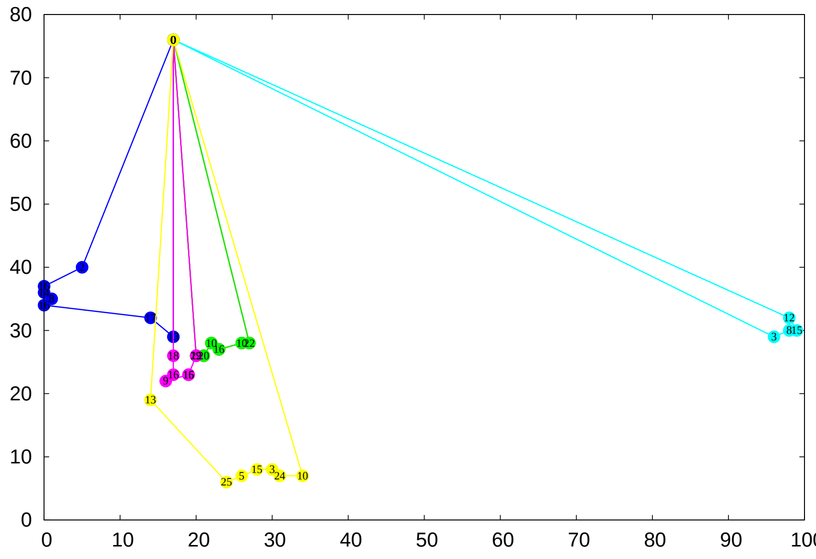
A-n69-k9



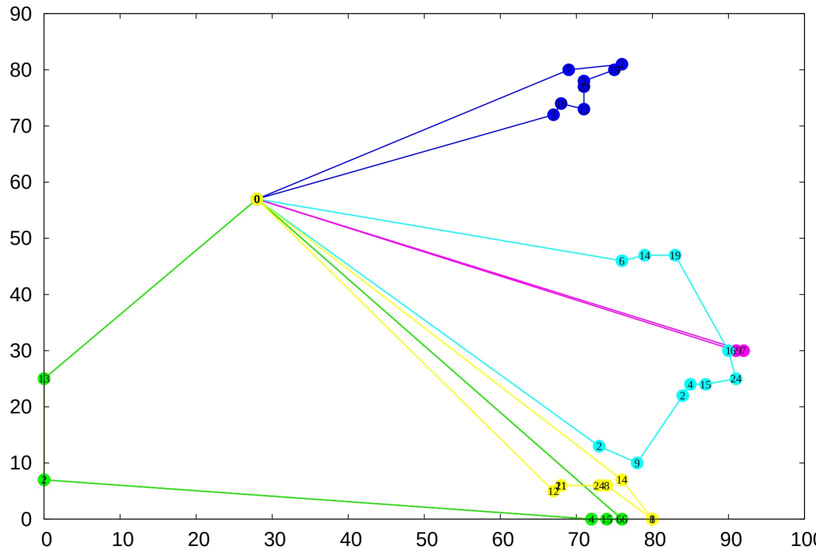
A-n80-k10



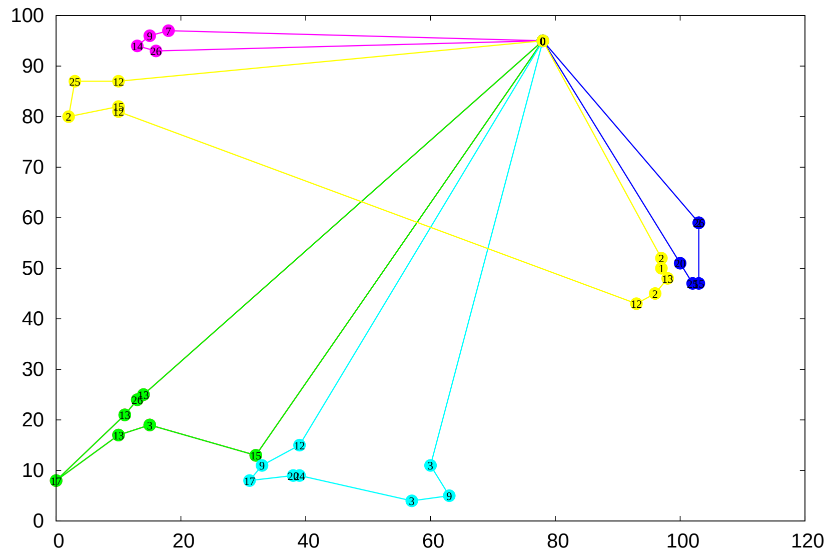
B-n31-k5



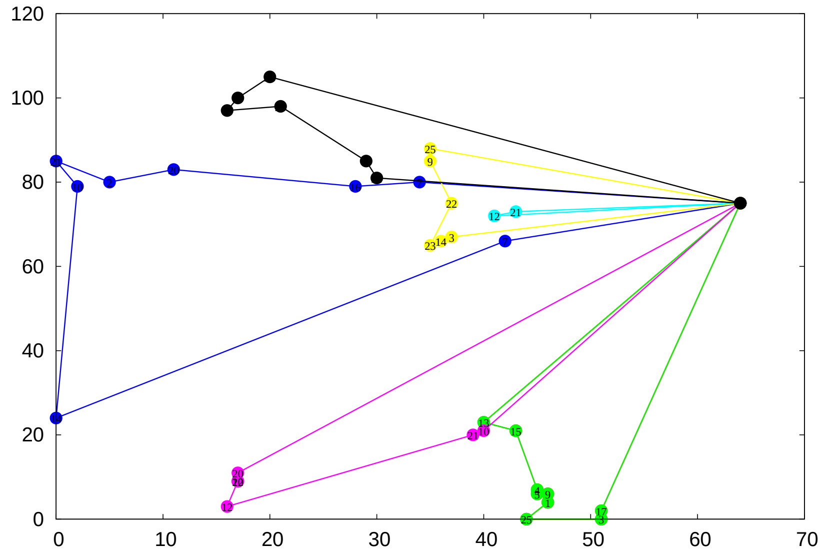
B-n34-k5



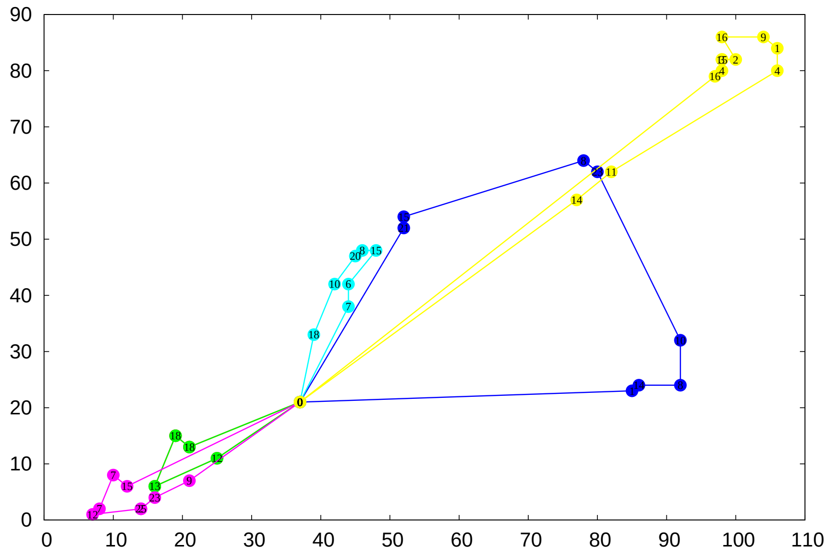
B-n35-k5



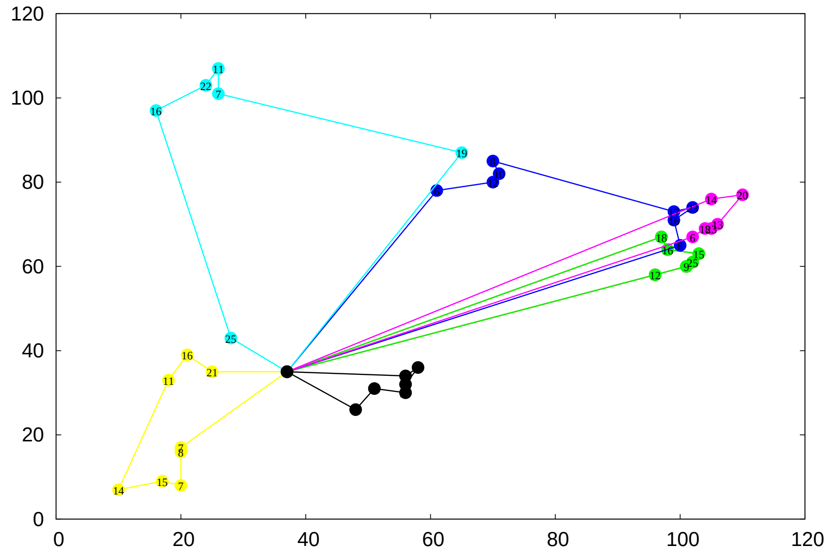
B-n38-k6



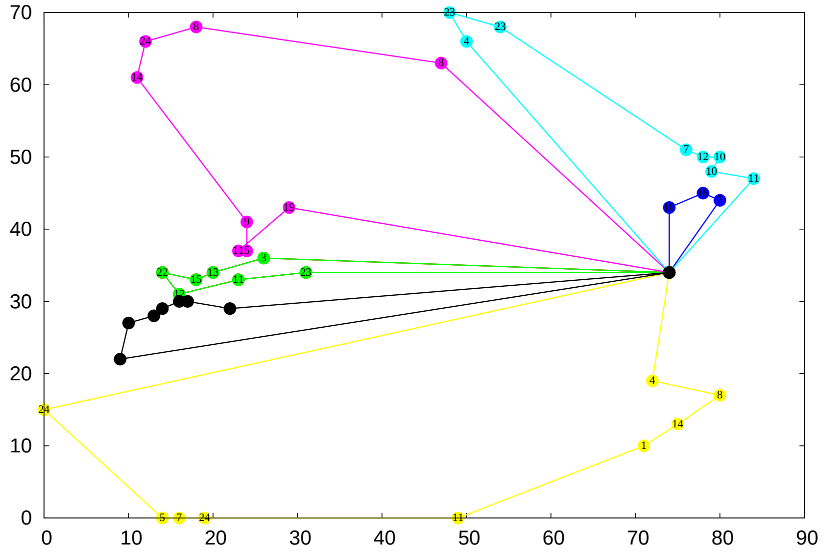
B-n39-k5

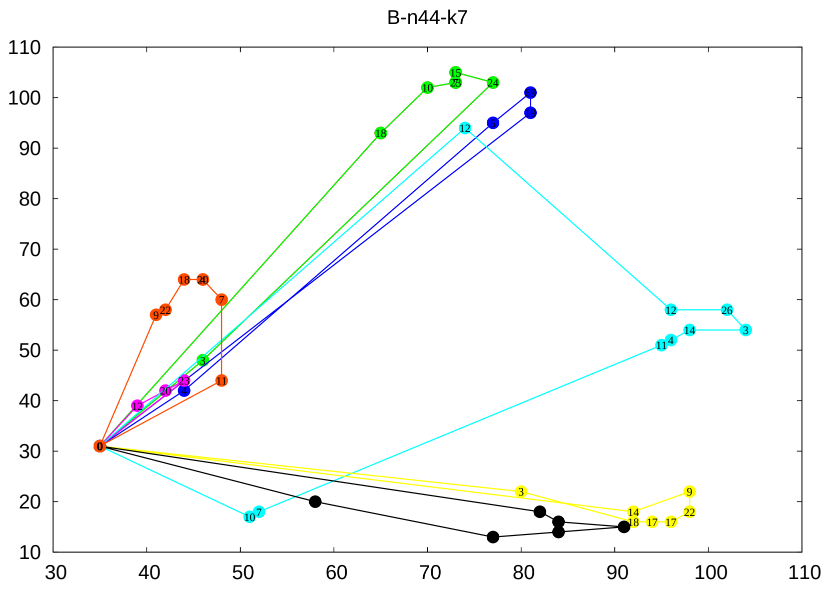


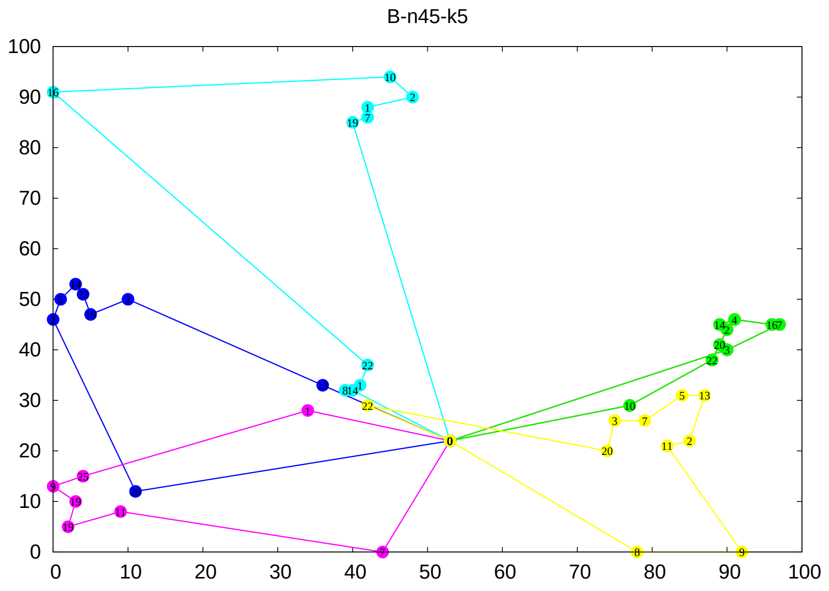
B-n41-k6



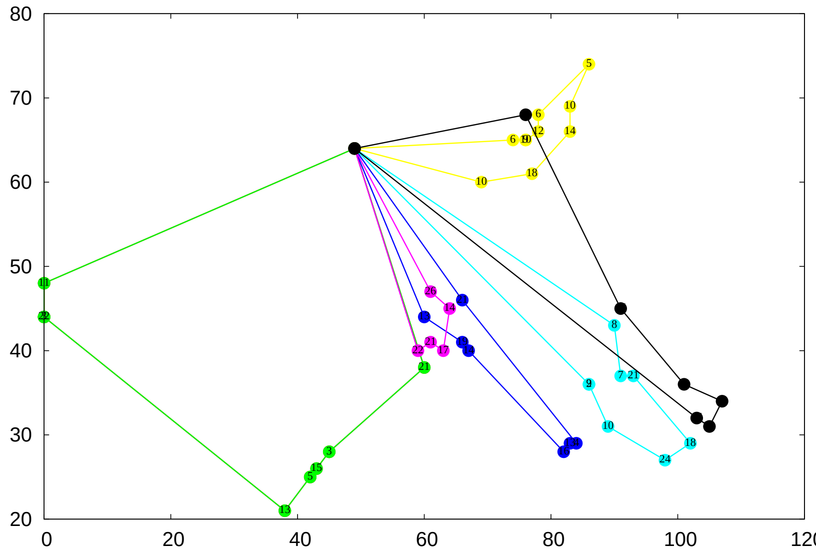
B-n43-k6



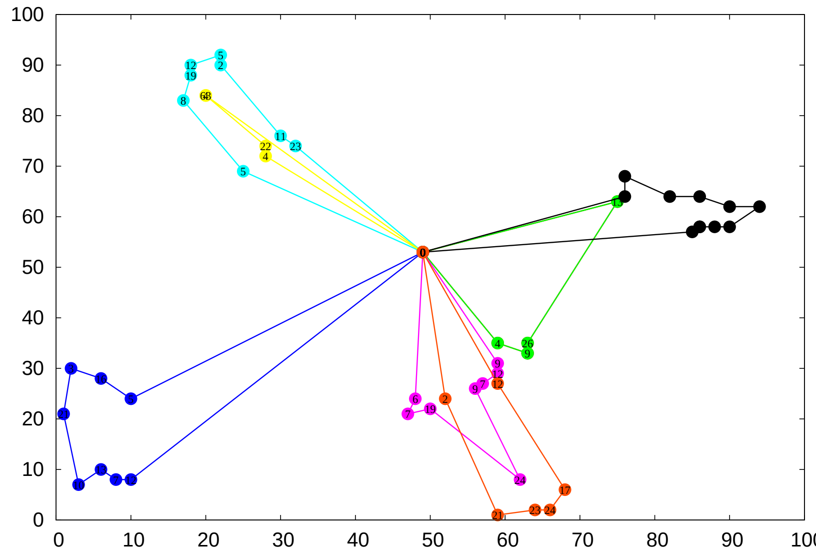

B-n44-k7


B-n45-k5

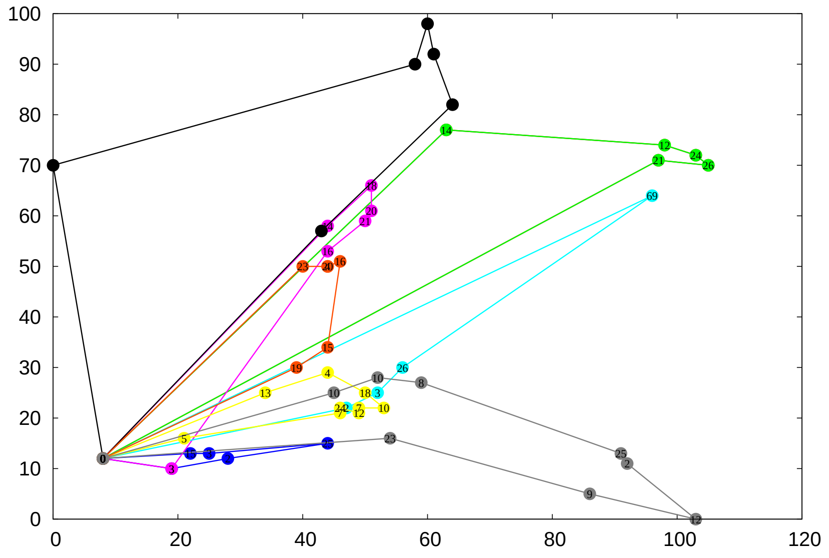
B-n45-k6



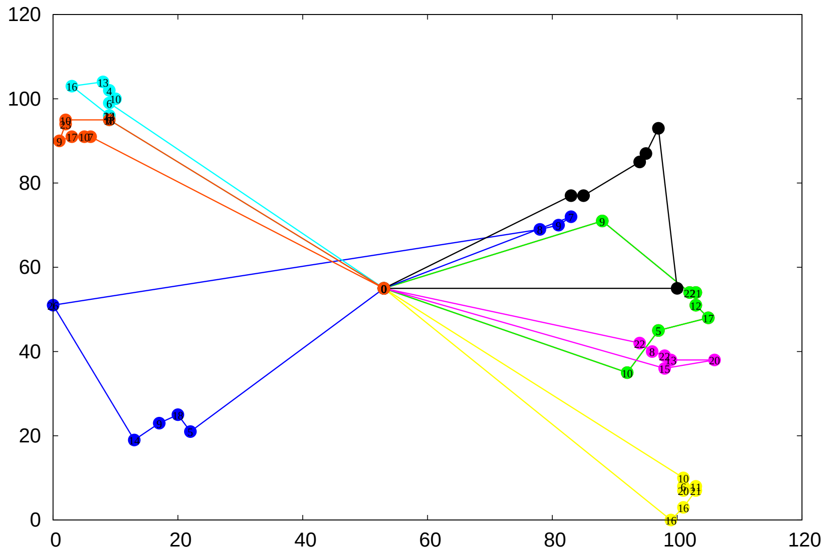
B-n50-k7



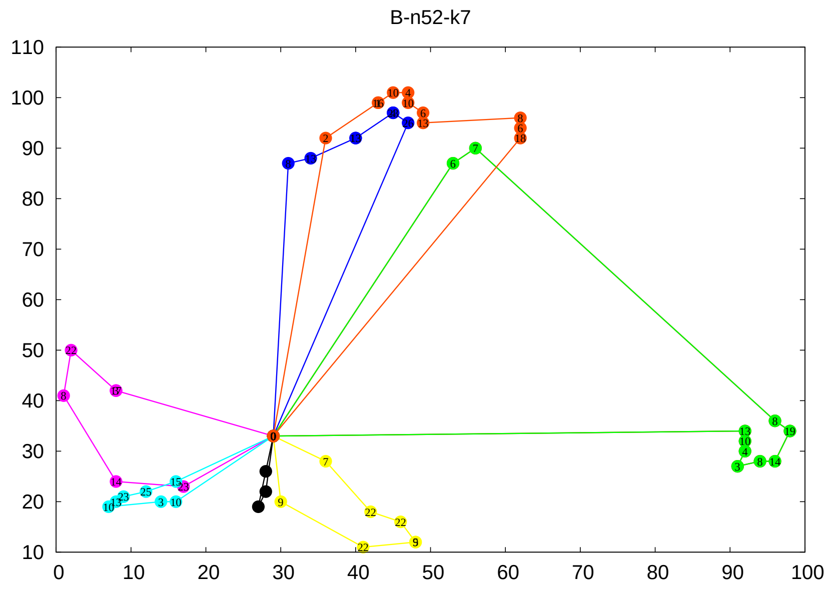
B-n50-k8



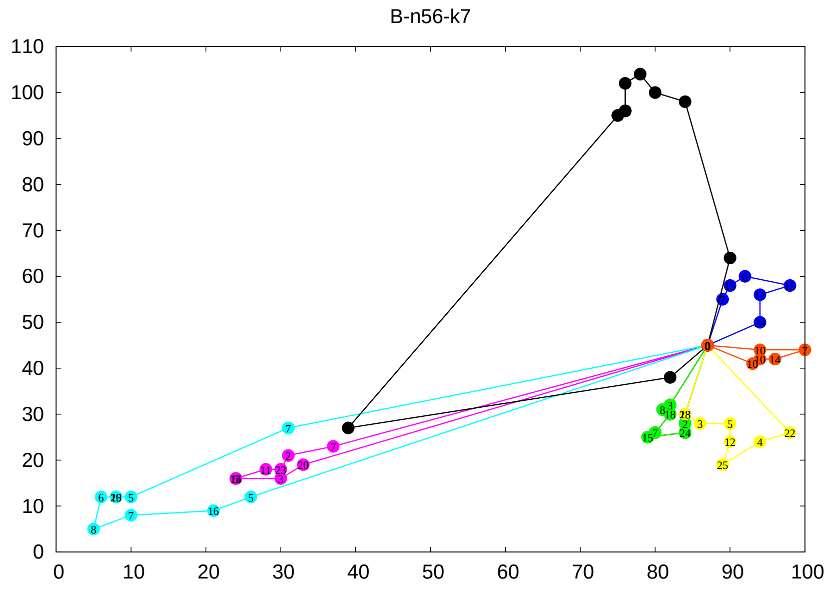
B-n51-k7



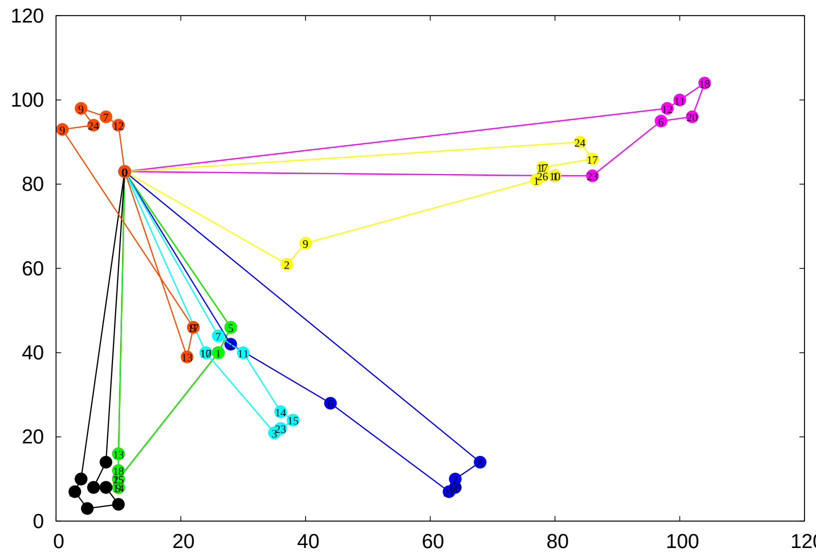
B-n52-k7



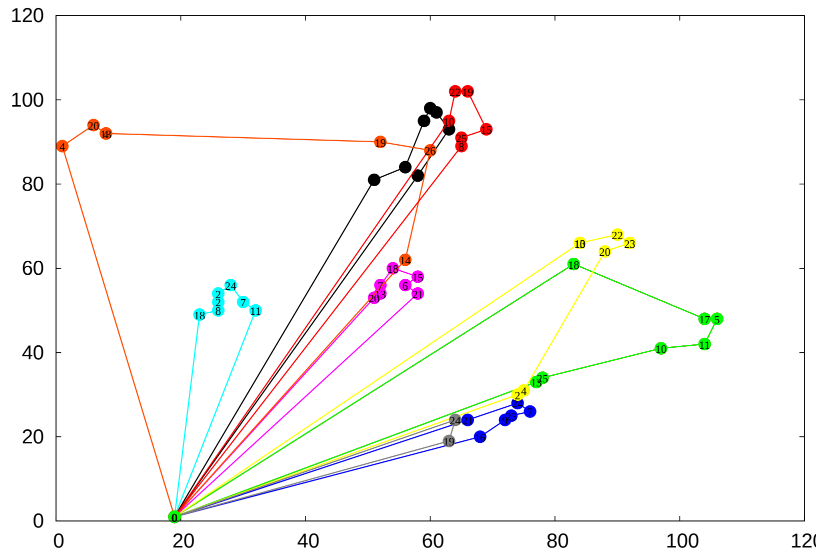
B-n56-k7



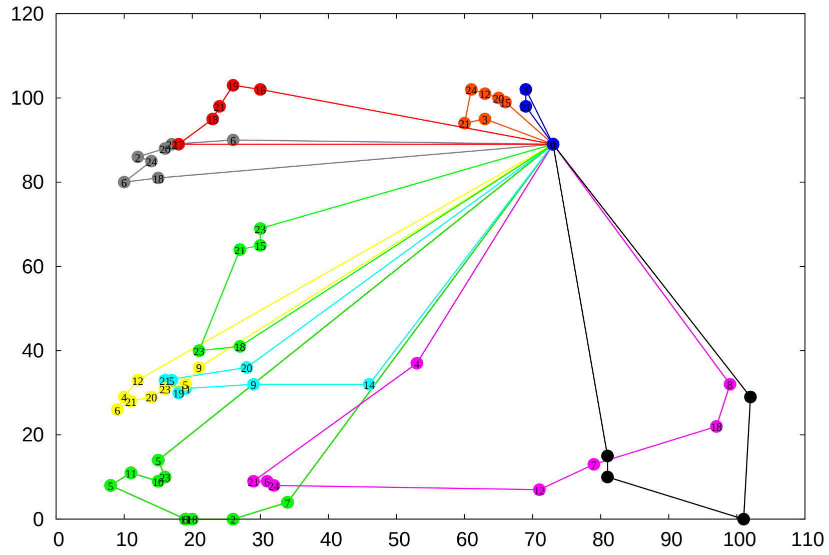
B-n57-k7



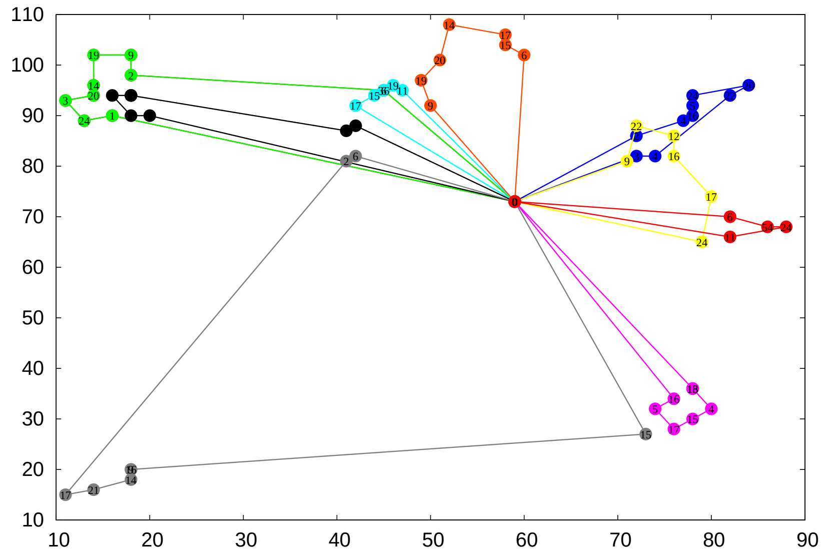
B-n57-k9



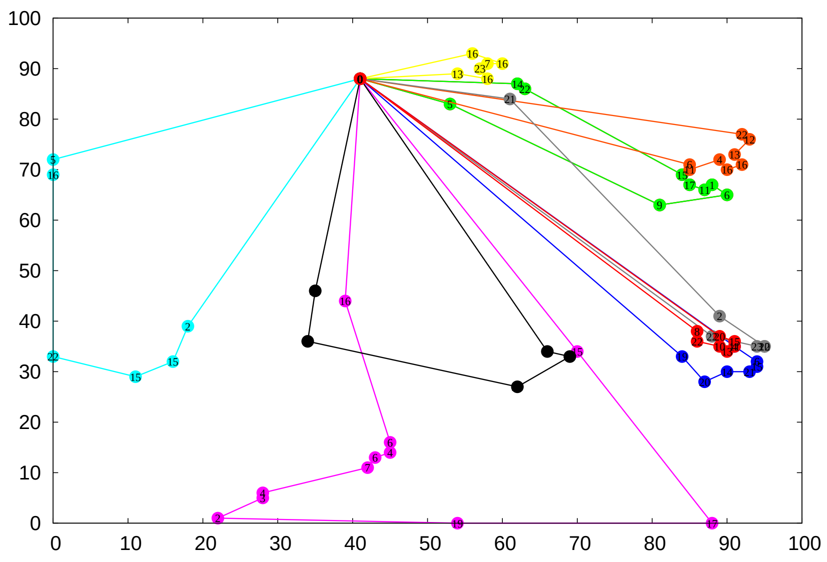
B-n63-k10



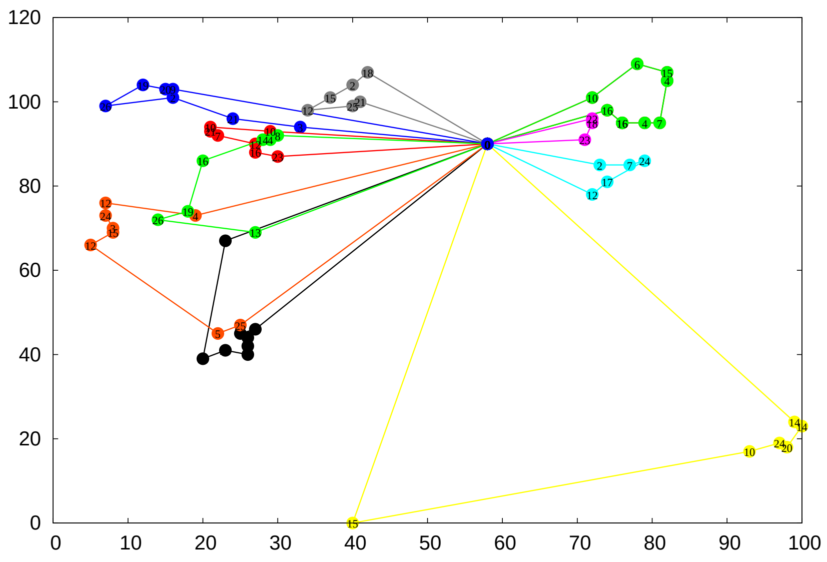
B-n64-k9



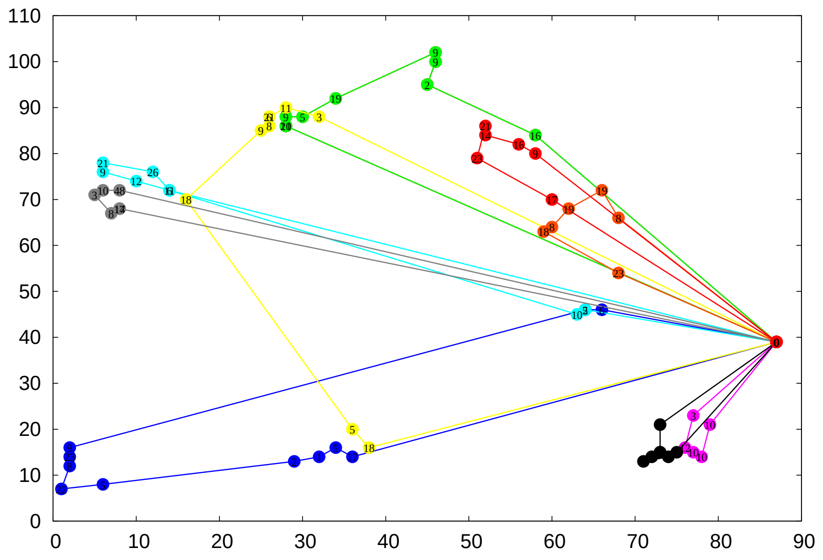
B-n66-k9



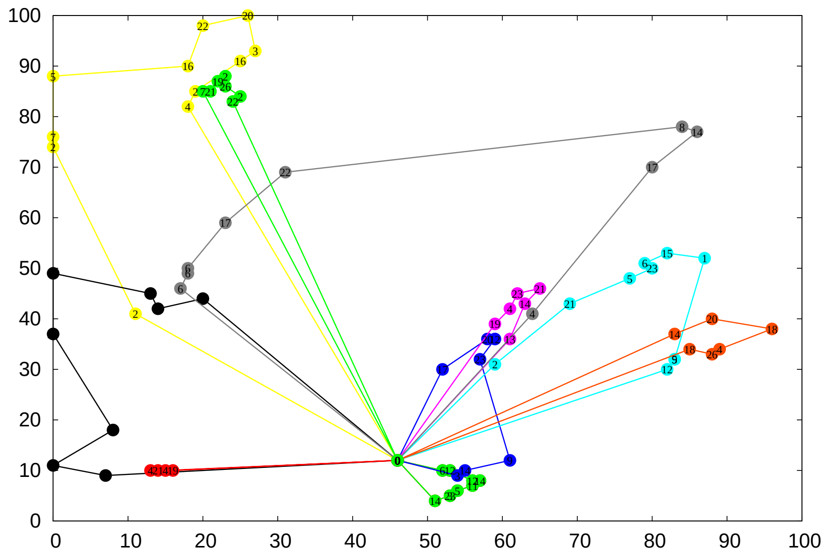
B-n67-k10

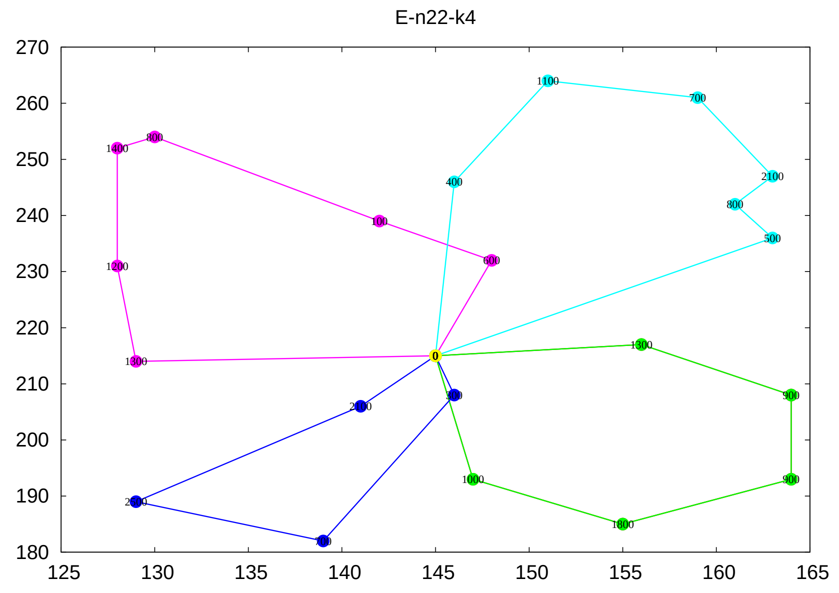


B-n68-k9

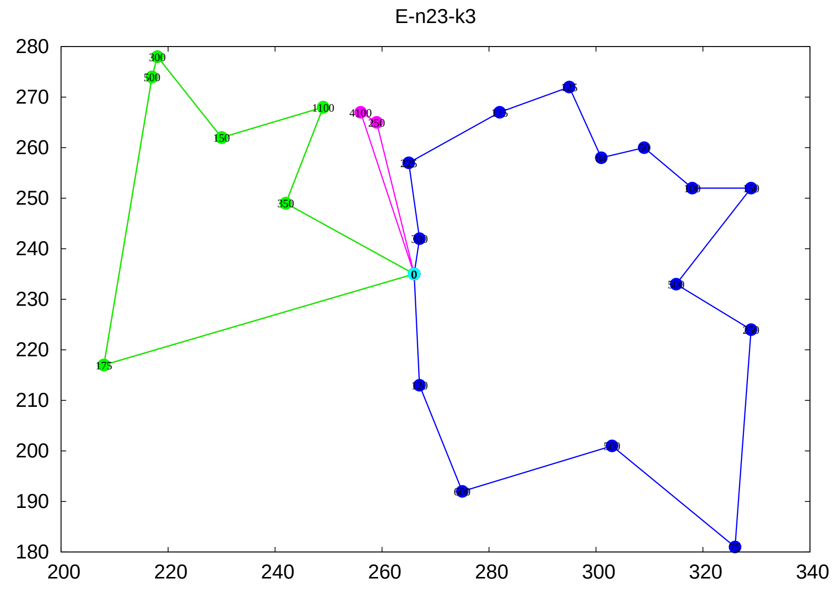


B-n78-k10



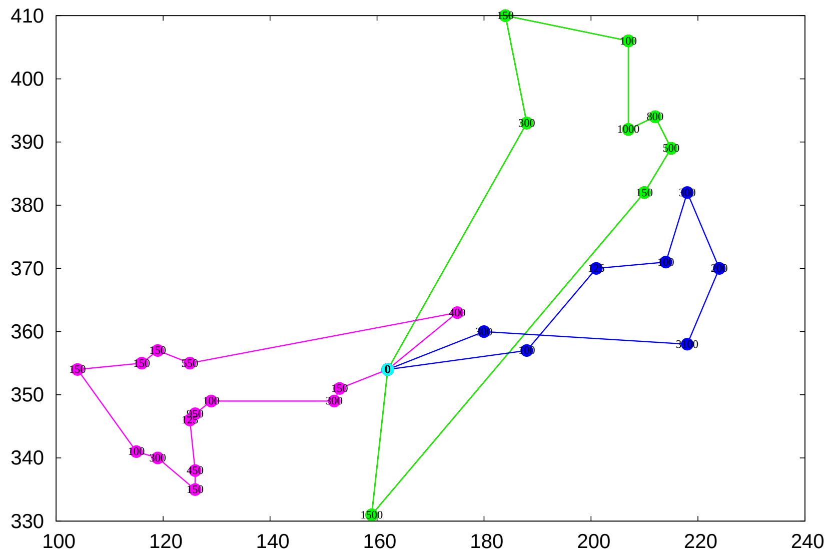


E-n22-k4

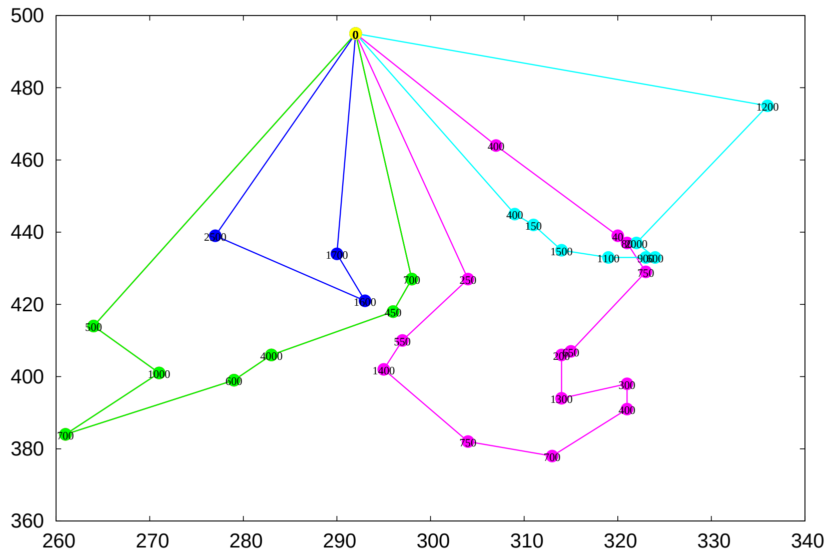


E-n23-k3

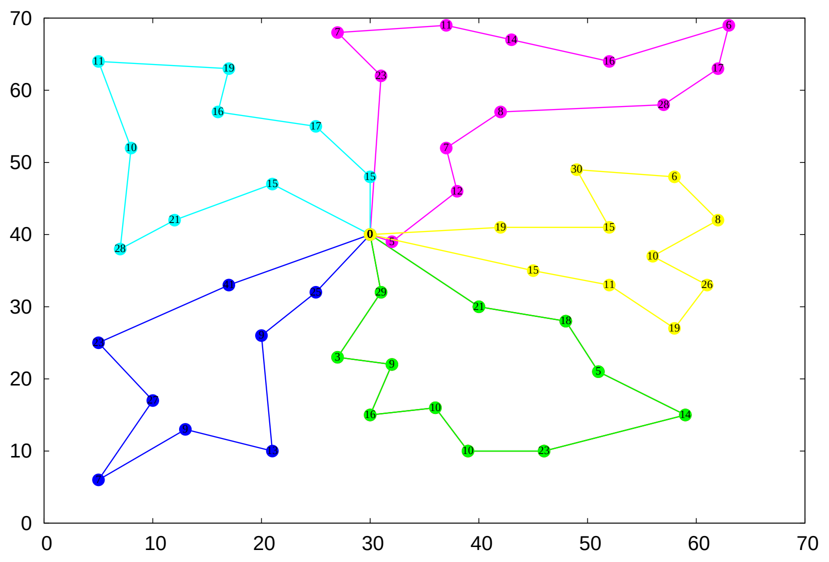
E-n30-k3



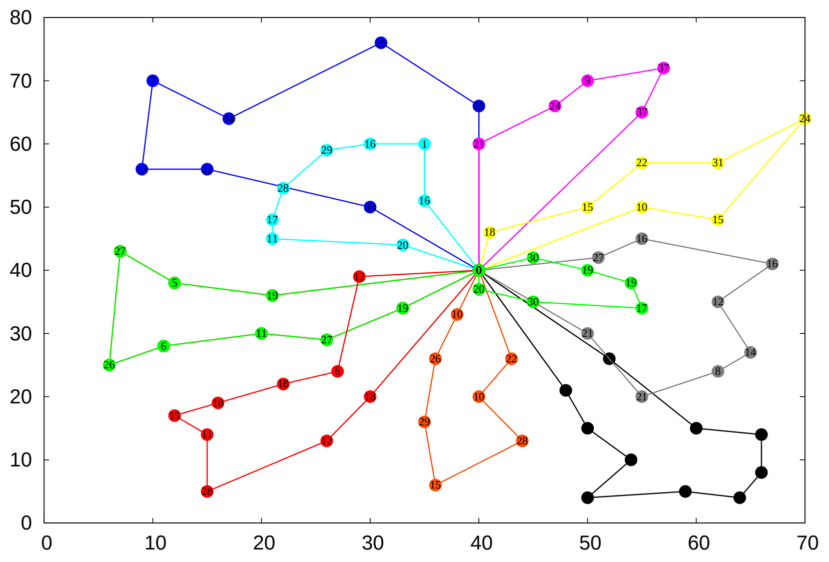
E-n33-k4



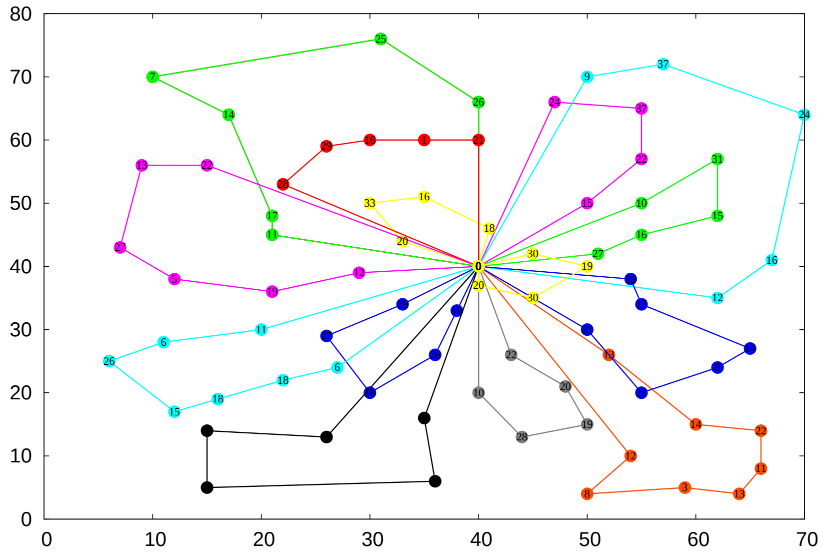
E-n51-k5



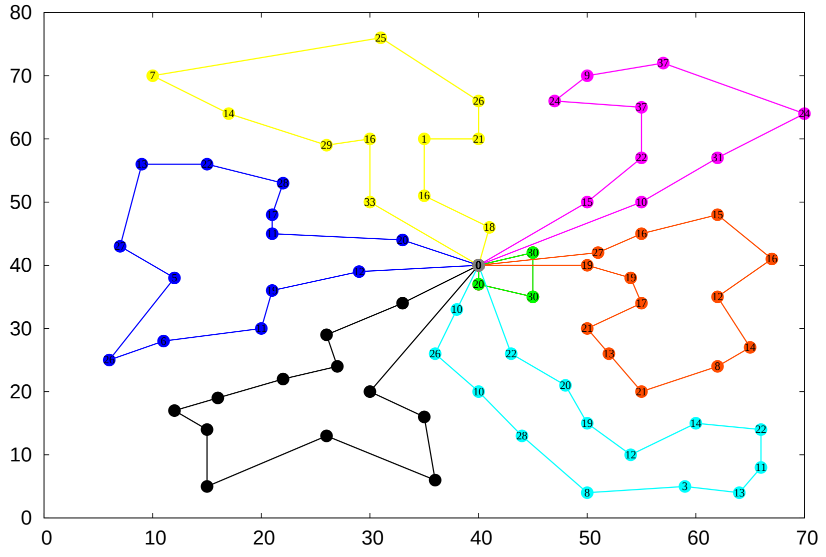
E-n76-k10



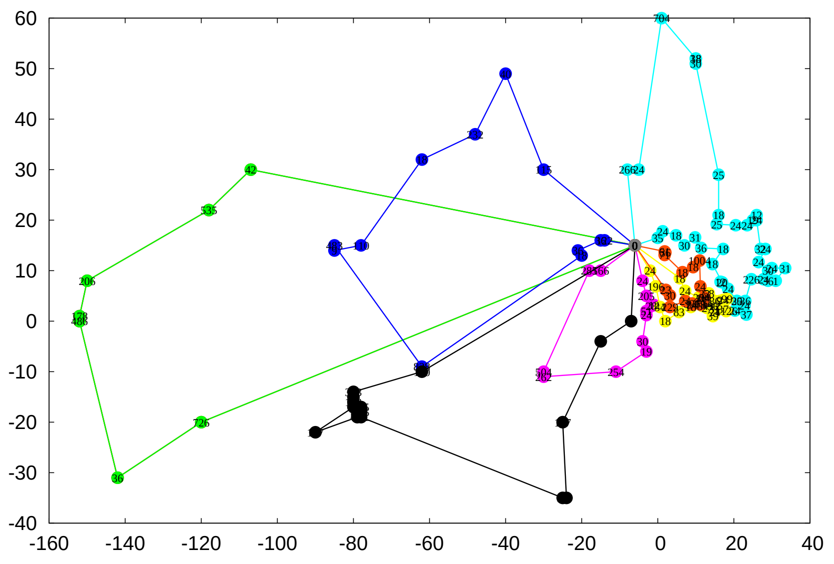
E-n76-k14



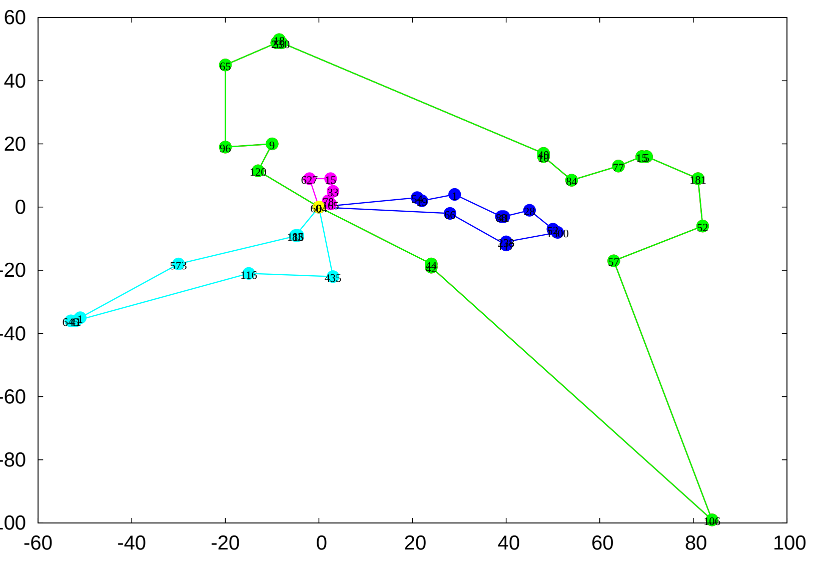
E-n76-k7



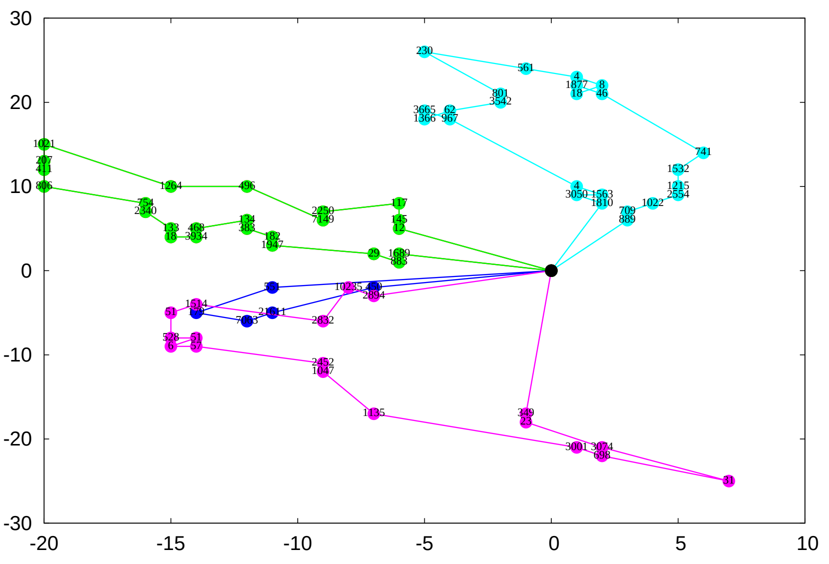
F-n135-k7



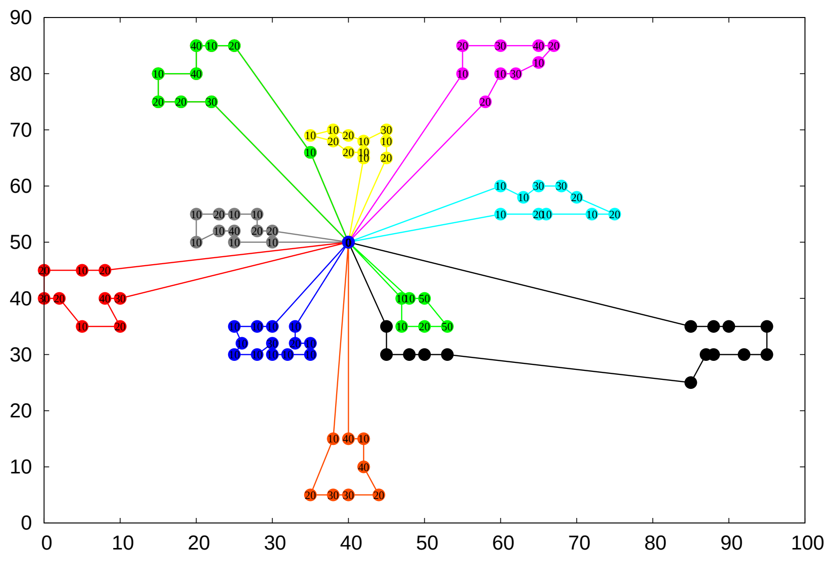
F-n45-k4



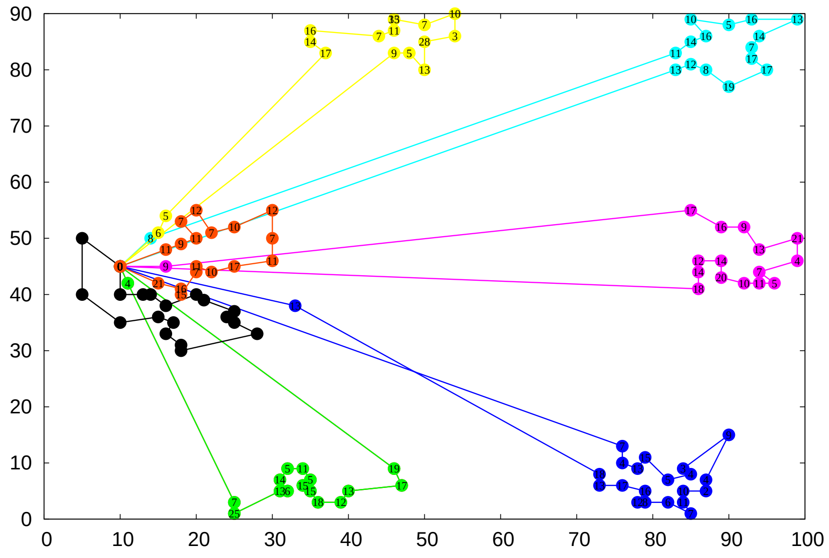
F-n72-k4



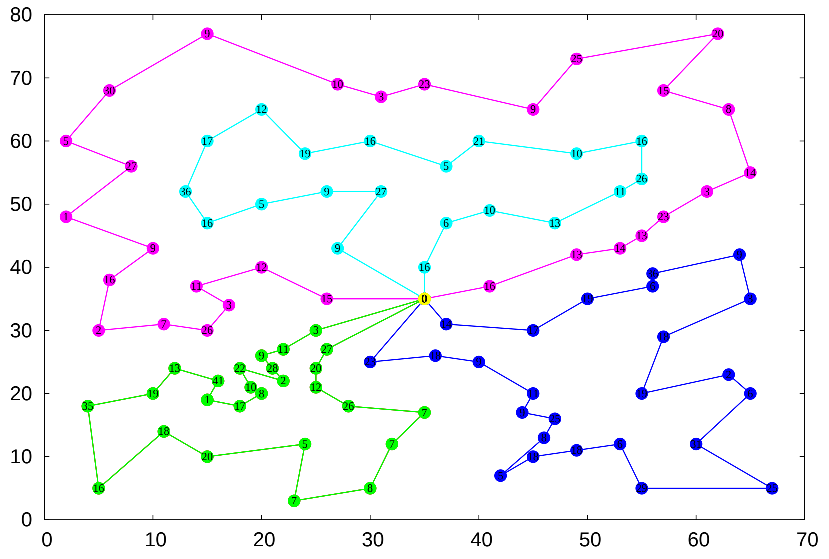
M-n101-k10

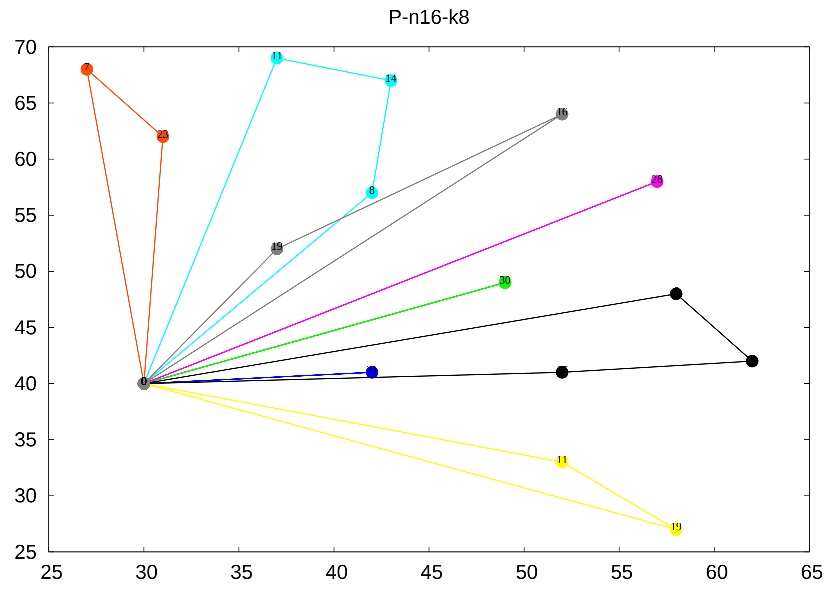


M-n121-k7

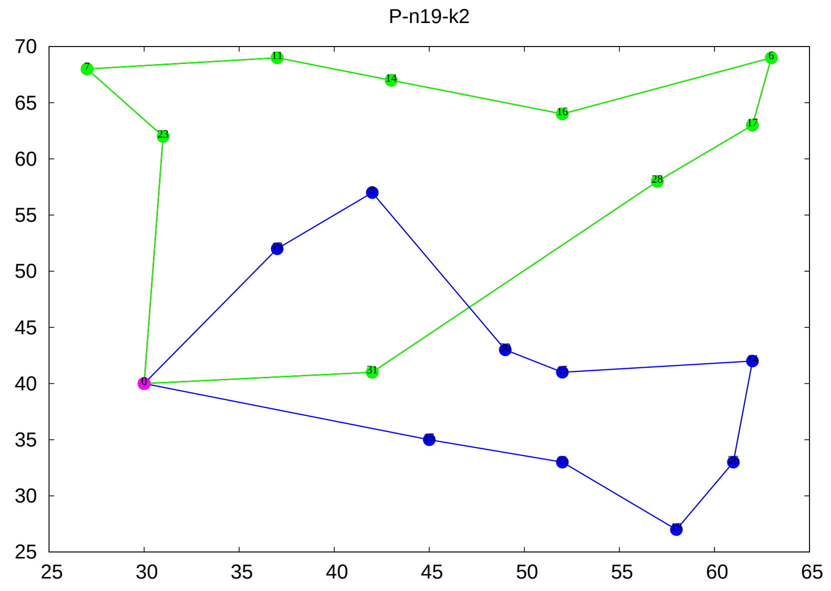


P-n101-k4



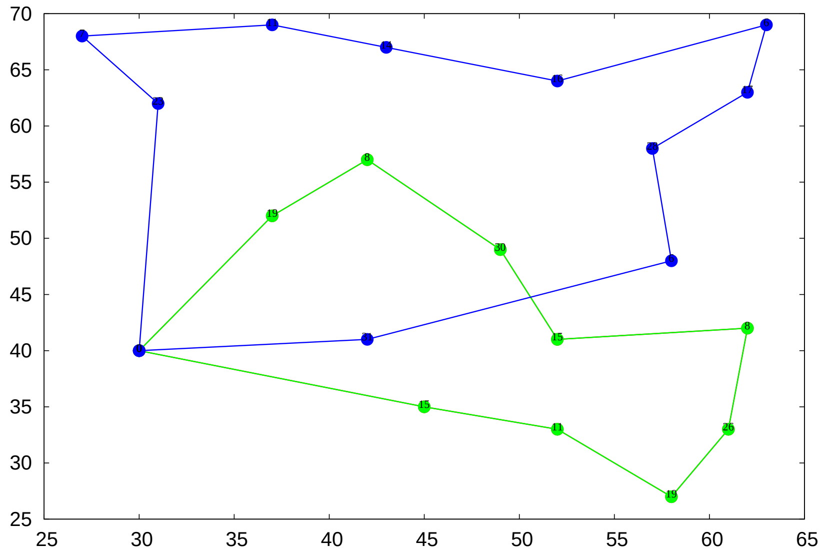


P-n16-k8
7
23
11
14
16
8
19
28
30
0
11
19

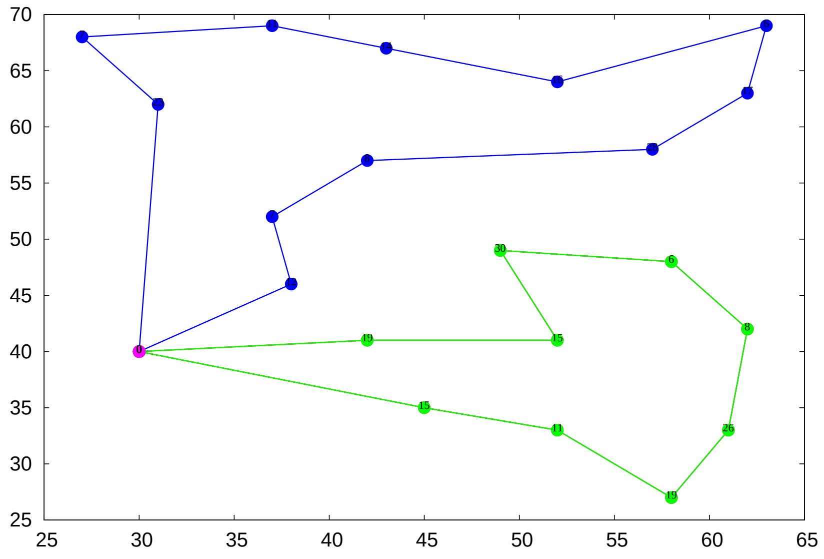


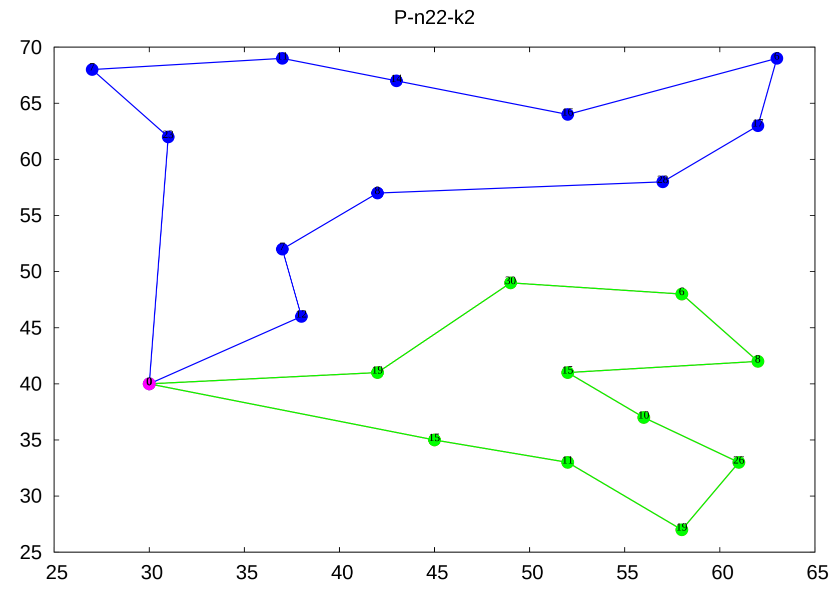
P-n19-k2
7
11
14
16
6
23
17
28
0
31

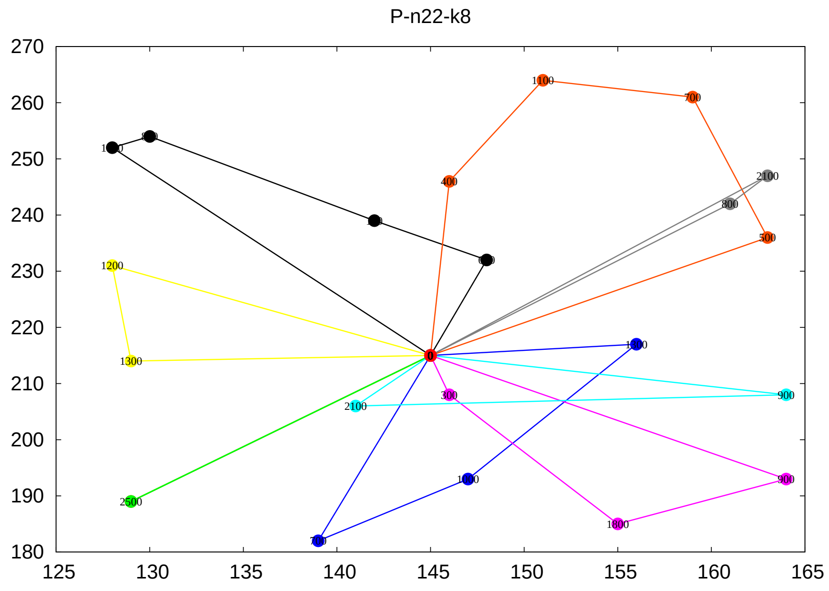
P-n20-k2



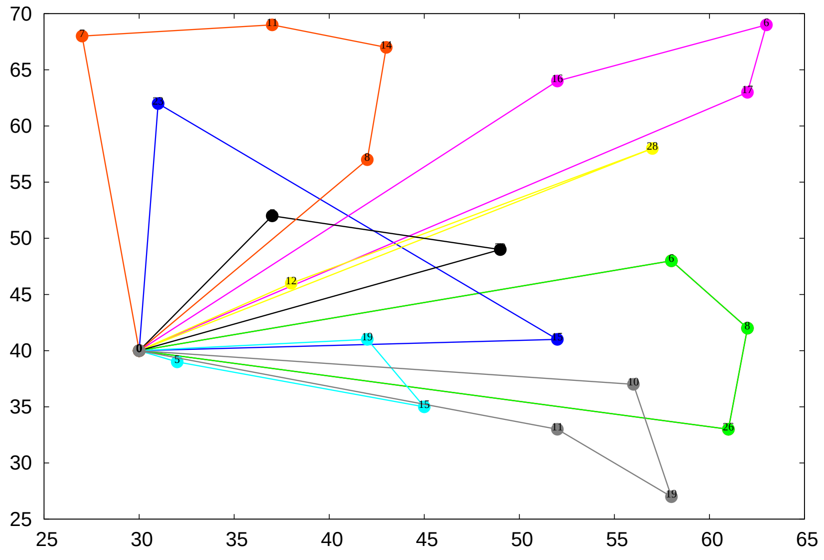
P-n21-k2



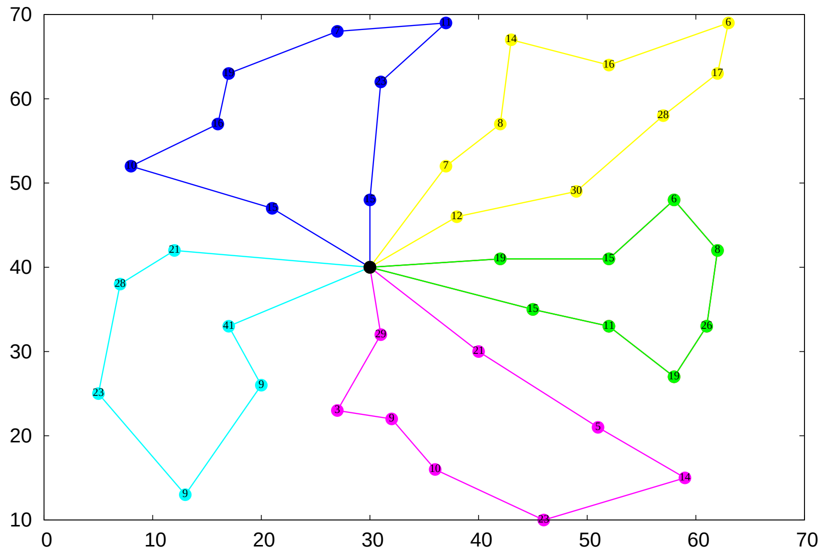

P-n22-k2
70
65
60
55
50
45
40
35
30
25
25
30
35
40
45
50
55
60
65
0
19
30
6
8
15
10
26
15
11
19


P-n22-k8
270
260
250
240
230
220
210
200
190
180
125
130
135
140
145
150
155
160
165
1100
700
400
2100
800
500
1200
1300
300
900
2100
900
2500
1800

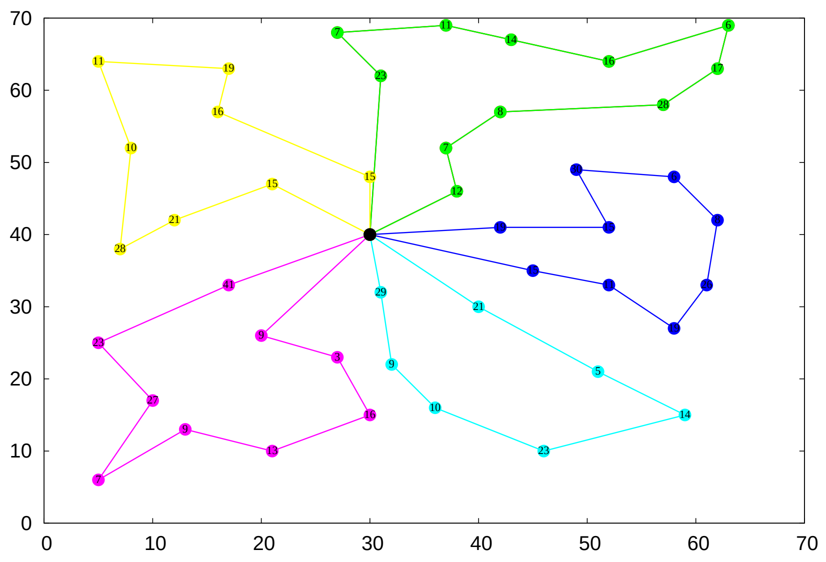
P-n23-k8



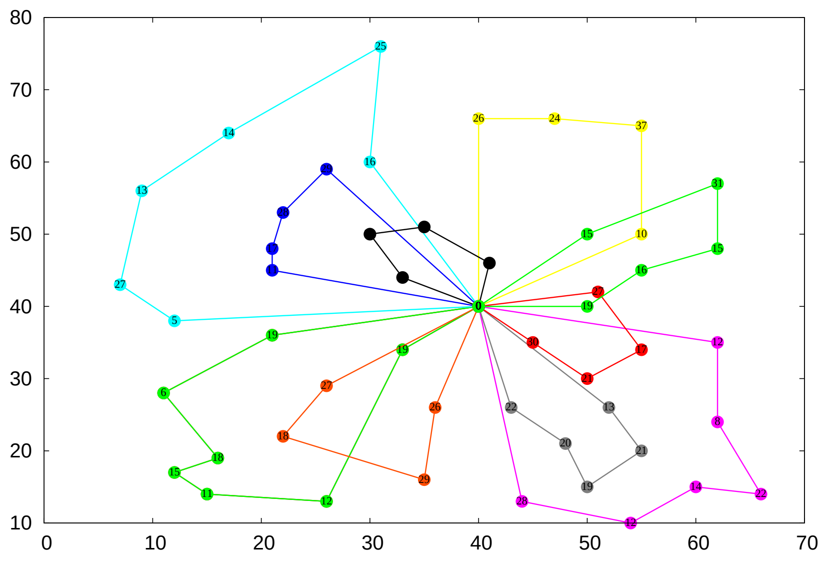
P-n40-k5



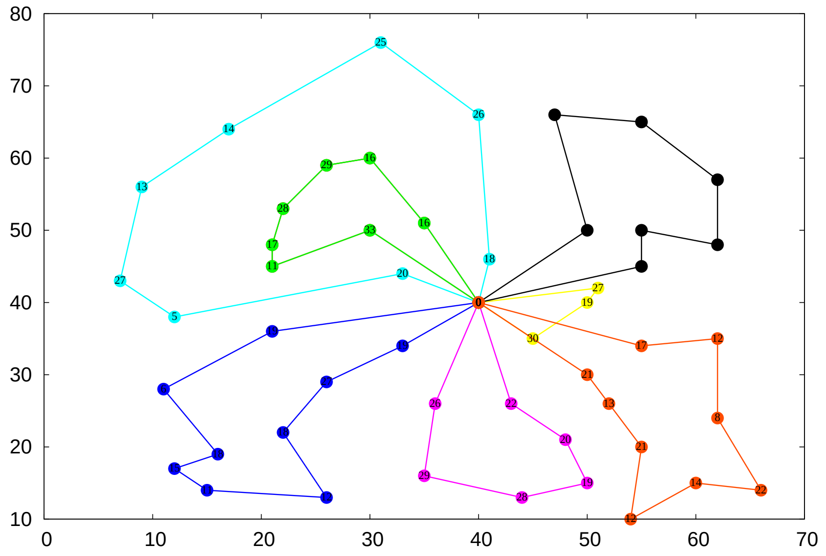
P-n45-k5



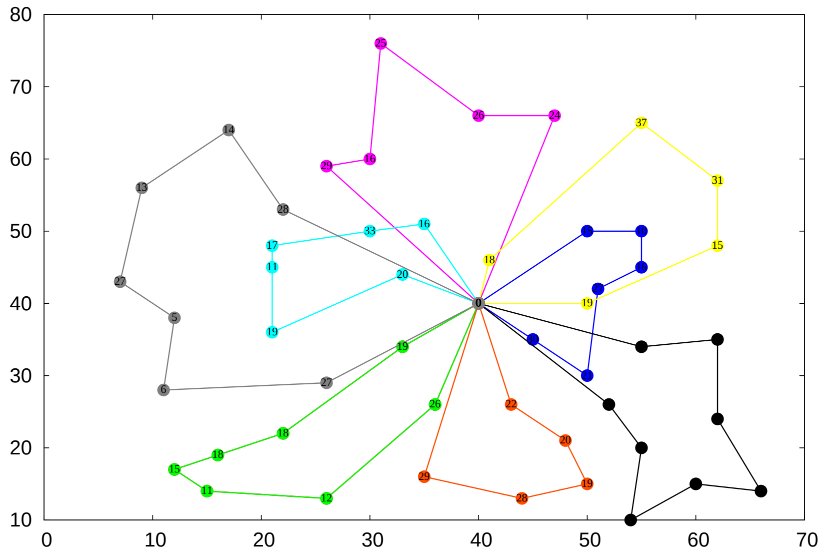
P-n50-k10



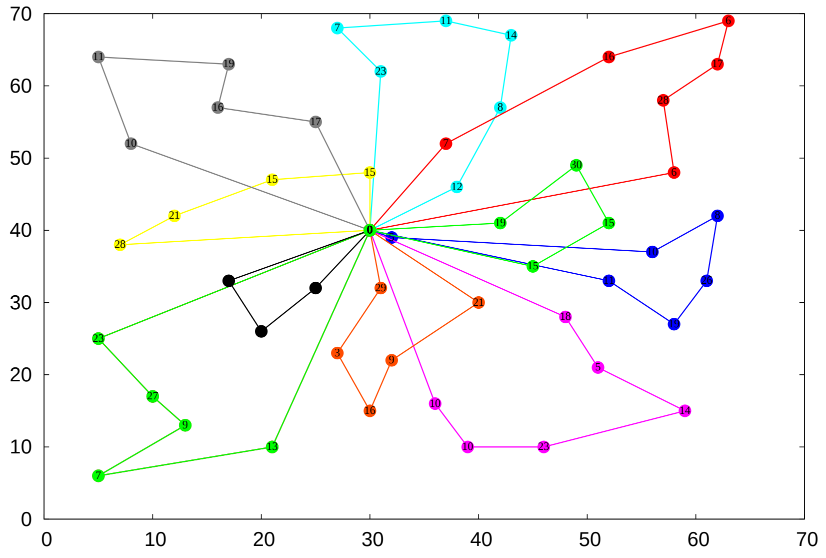
P-n50-k7



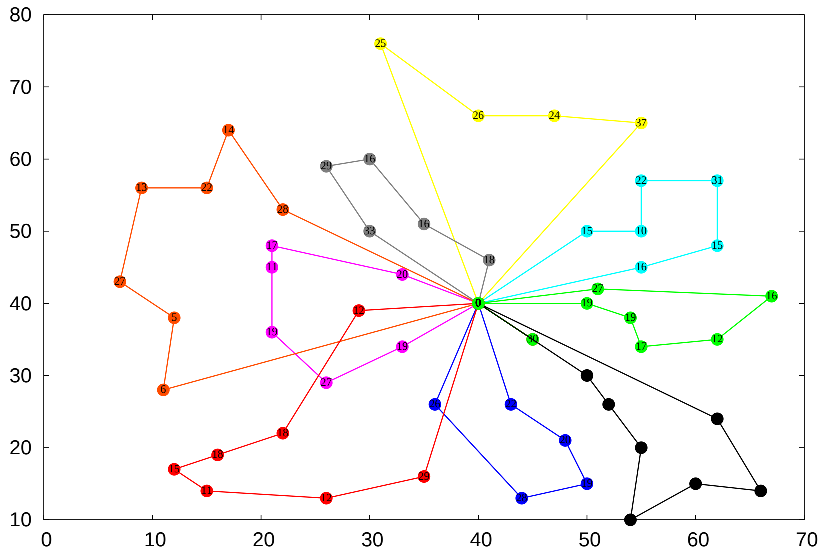
P-n50-k8

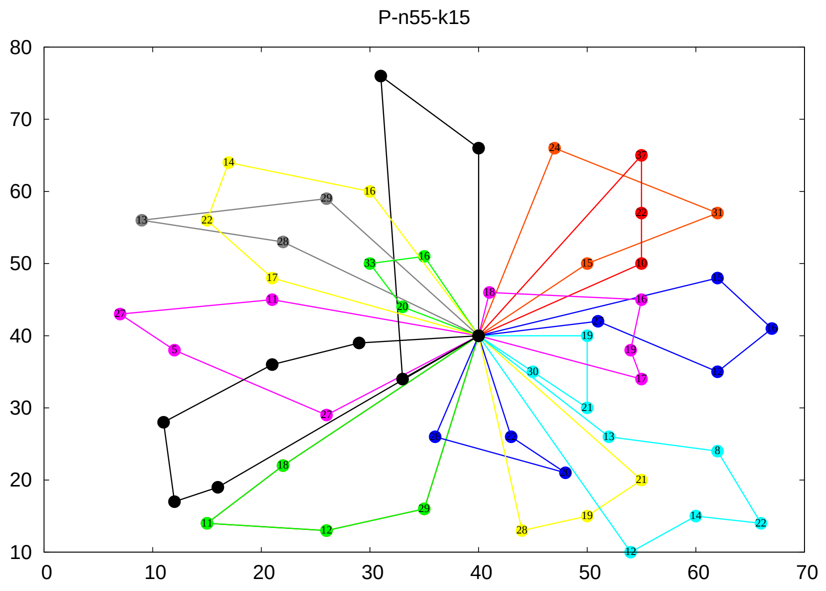


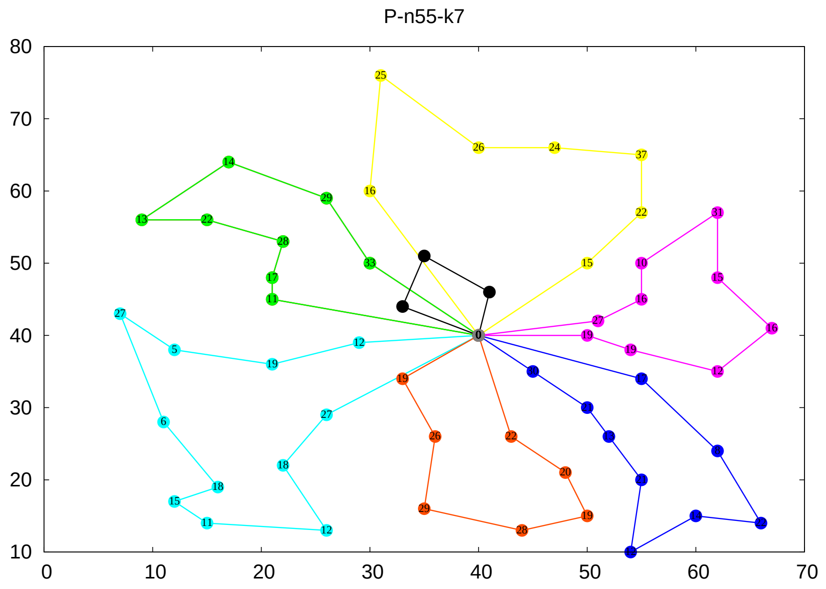
P-n51-k10



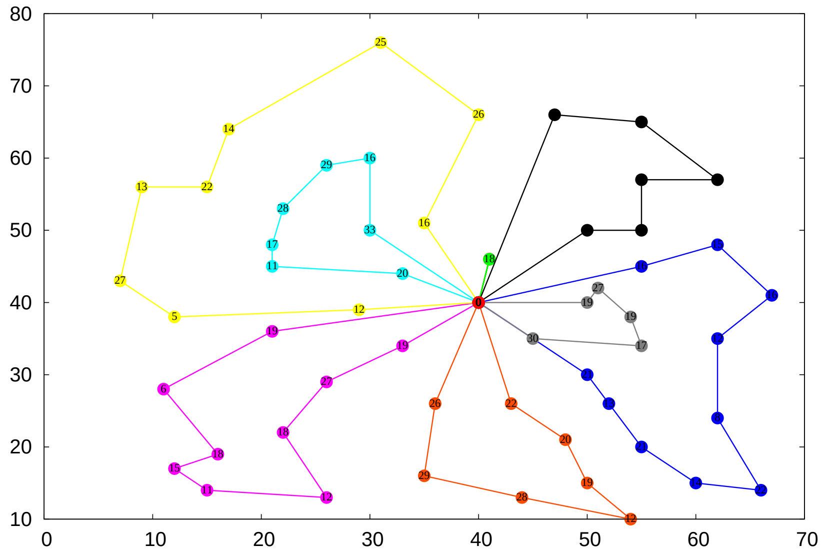
P-n55-k10



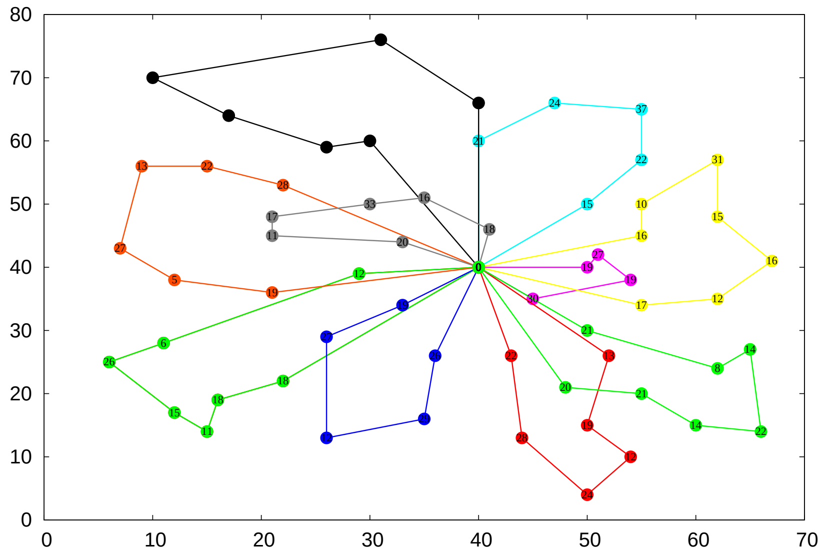

P-n55-k15

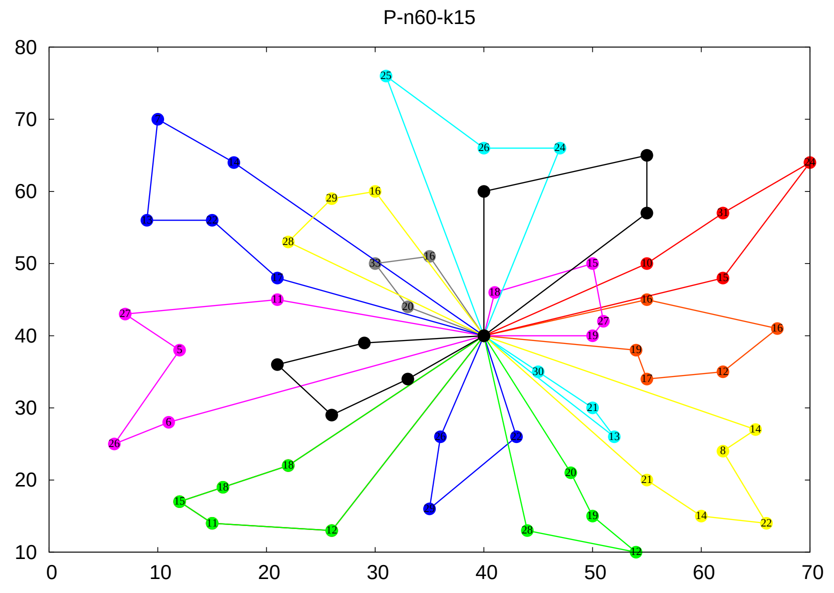

P-n55-k7

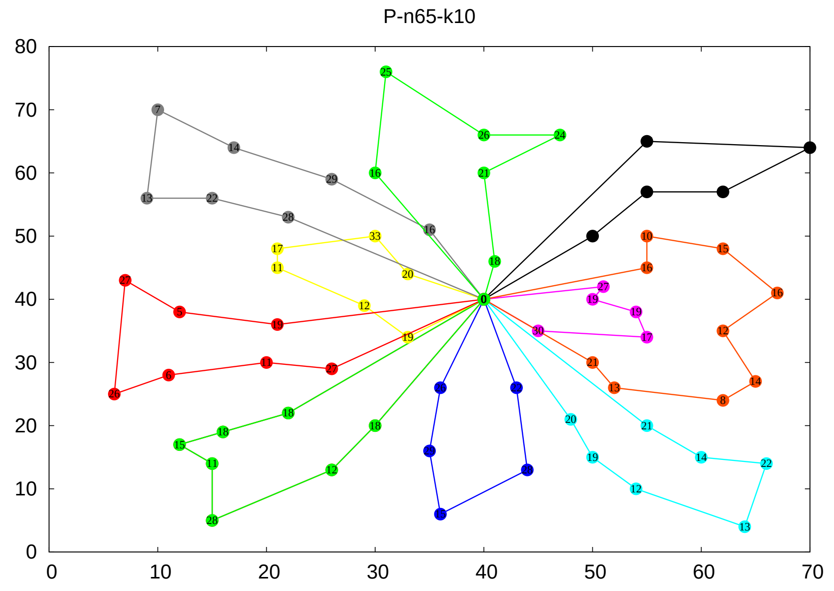
P-n55-k8



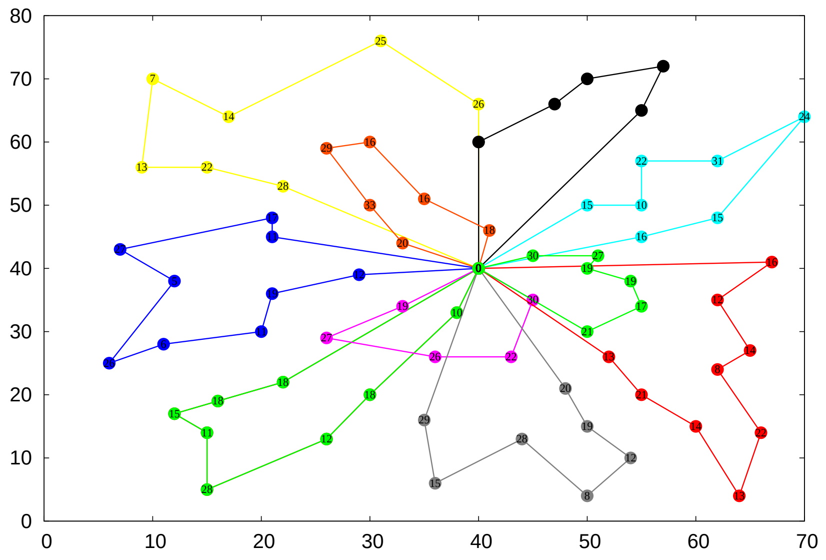
P-n60-k10



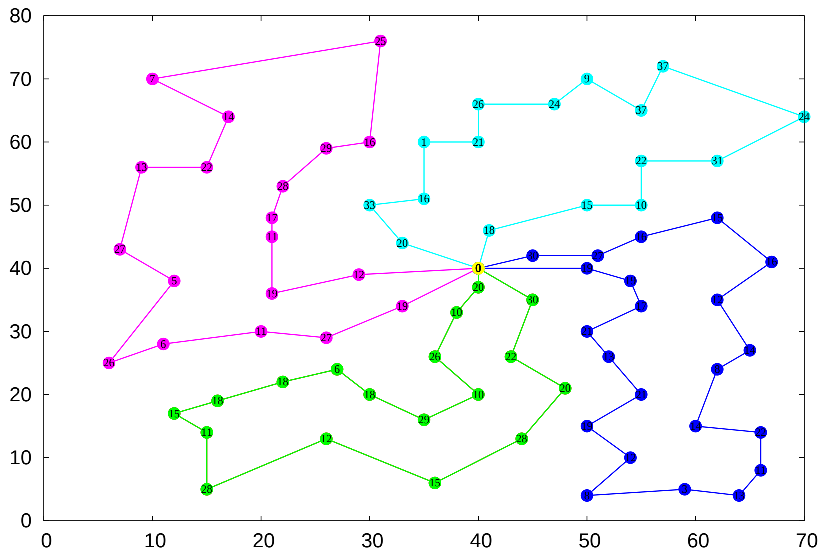

P-n60-k15
0
10
20
30
40
50
60
70
80


P-n65-k10
0
10
20
30
40
50
60
70
80

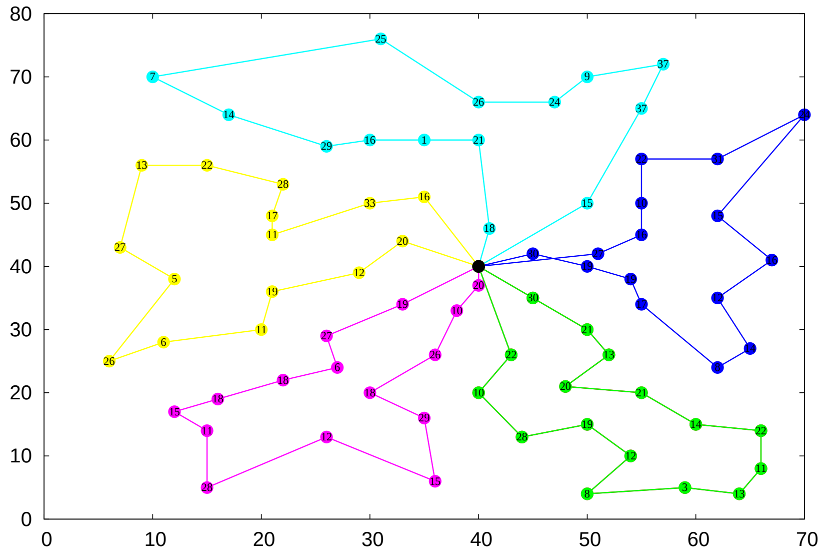
P-n70-k10



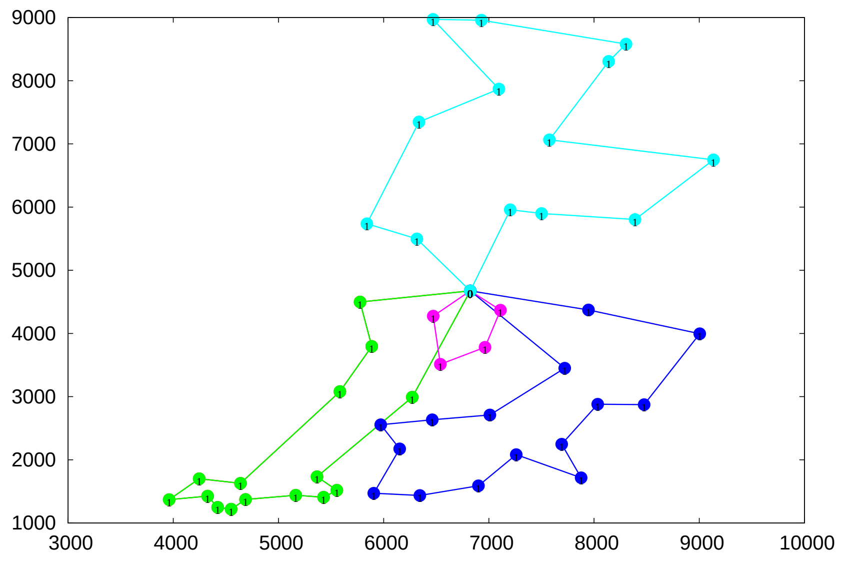
P-n76-k4



P-n76-k5



att-n48-k4



Appendix C. Script to Graph a Problem in TSPLIB Format with GnuPlot

```
#!/usr/bin/perl


if ($#ARGV != 0) {
    die "Usage: %readProb.pl <vrp file>\n";
}



open DATAFILE, "$ARGV[0]"
        or die "Error opening file $ARGV[0]";


while (<DATAFILE>) {#processes first argument


        if (m/DIMENSION.*/) {
            @fields = split(" ", $_);
            $num_locations = $fields[2];
        }


        if (m/NODE_COORD_SECTION/) {
                        $x_max=0;
                        $y_max=0;
                        $x_min=0;
                        $y_min=0;


            for ($i=1; $i <= $num_locations; $i+=1) {
                $_ = <DATAFILE>;
                #first location is depot


                @fields = split(" ", $_);


                $x_coords[$i] = $fields[1];
                $y_coords[$i] = $fields[2];


                                if ($x_coords[$i] > $x_max) {
                                        $x_max = $x_coords[$i];
                                }
                                if ($y_coords[$i] > $y_max) {
                                        $y_max = $y_coords[$i];
                                }
                                if ($x_coords[$i] < $x_min) {
                                        $x_min = $x_coords[$i];
                                }
                                if ($y_coords[$i] < $y_min) {
                                        $y_min = $y_coords[$i];
                                }
            }
                }
            if (m/DEMAND_SECTION/) {
        for ($i=1; $i <= $num_locations; $i+=1) {
            $_ = <DATAFILE>;
            #first location is depot


            @fields = split(" ", $_);
            $demands[$i] = $fields[1];
        }


        }


}
close DATAFILE;


open(outfile, ">/tmp/tempdepot");
for ($i=1; $i <= $num_locations; $i +=1) {
        print outfile "$x_coords[$i] $y_coords[$i]\n";
}
close outfile;


$title = substr($ARGV[0], 3, (length $ARGV[0])-7);
open(outfile, ">/tmp/temp.gnu");
print outfile "set term postscript eps color solid 25\n";
print outfile "set nokey\n";
print outfile "set size 1.5,1.5\n";
print outfile "set pointsize 4.0\n";
print outfile "set out \"/dev/null\"\n";
print outfile "set title \"$title\"\n";
$x_max +=1;
$y_max +=1;
$y_min -=1;
$x_min -=1;
print outfile "plot [$x_min:$x_max] [$y_min:$y_max] \"/tmp/tempdepot\" using 1:2 with points\n";


for ($i=1; $i <= $num_locations; $i+=1) {
        $temp = $y_coords[$i] +0.5;
        print outfile "set label \"$demands[$i]\" at first $x_coords[$i],'';
        print outfile '' first $temp font \"Times,14\" front center\n";
}


print outfile "set out \"$title-prob.eps\"\n";
print outfile "replot\n";
close outfile;


system "gnuplot /tmp/temp.gnu";
```

Appendix D. Script to to Produce vrpPlot.awk Format

```
#!/usr/bin/perl


#read the file containing the X Y values in TSPlib format

#these values are contained between NODE_COORD_SECTION and

#DEMAND_SECTION in the following form: id x y


if ($#ARGV != 1) {

        die "Usage: %readOptimal.pl <vrp file> <opt file>\n";

}




open DATAFILE, "$ARGV[0]"

        or die "Error opening file $ARGV[0]";



while (<DATAFILE>) {#processes first argument


        if (m/DIMENSION.*/) {

                @fields = split(" ", $_);

                $num_locations = $fields[2];

        }




        if (m/NODE_COORD_SECTION/) {

                for ($i=1; $i <= $num_locations; $i+=1) {

                        $_ = <DATAFILE>;

                        #first location is depot


                        @fields = split(" ", $_);


                        $x_coords[$i] = $fields[1];

                        $y_coords[$i] = $fields[2];

                }

        }


        if (m/DEMAND_SECTION/) {

                for ($i=1; $i <= $num_locations; $i+=1) {

                        $_ = <DATAFILE>;

                        #first location is depot


                        @fields = split(" ", $_);

                        $demands[$i] = $fields[1];

                }


        }


}

close DATAFILE;


#Sample file format for optimal values:


#Route #1: 21 31 19 17 13 7 26

#Route #2: 12 1 16 30

#Route #3: 27 24

#Route #4: 29 18 8 9 22 15 10 25 5 20

#Route #5: 14 28 11 4 23 3 2 6

#cost 784


#as each line is processed, spit out its coordinates

#on a line of their own

#separate routes with "* *" to include a final "* *" at

#EOF


open DATAFILE, "$ARGV[1]"

        or die "Error opening file $ARGV[1]";


while (<DATAFILE>) {

        @fields = split(" ", $_);


        $sum_of_demands = 0;


        if (!m/.*cost.*/) {

                print "$x_coords[1] $y_coords[1] $demands[1]\n";

                for ($i=2; $i<= $#fields; $i+=1) {

                        print "$x_coords[$fields[$i]+1] $y_coords[$fields[$i]+1]";

                    print " $demands[$fields[$i]+1]\n";

                        $sum_of_demands += $demands[$fields[$i]+1];

                }

                print "$x_coords[1] $y_coords[1] $demands[1]\n";

                print "*$sum_of_demands*\n";


        }

}
```

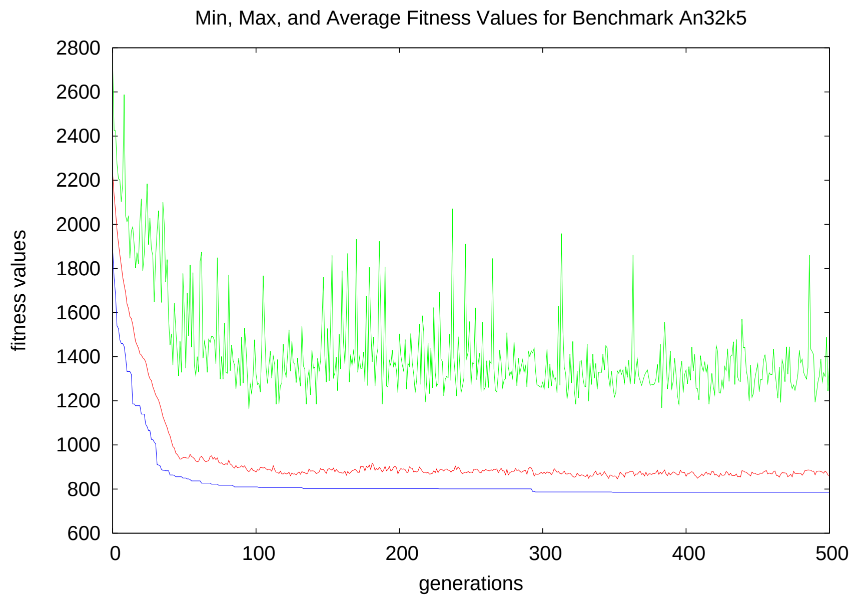
Appendix E. Script to Graph an Optimal Solution for a CVRP Problem

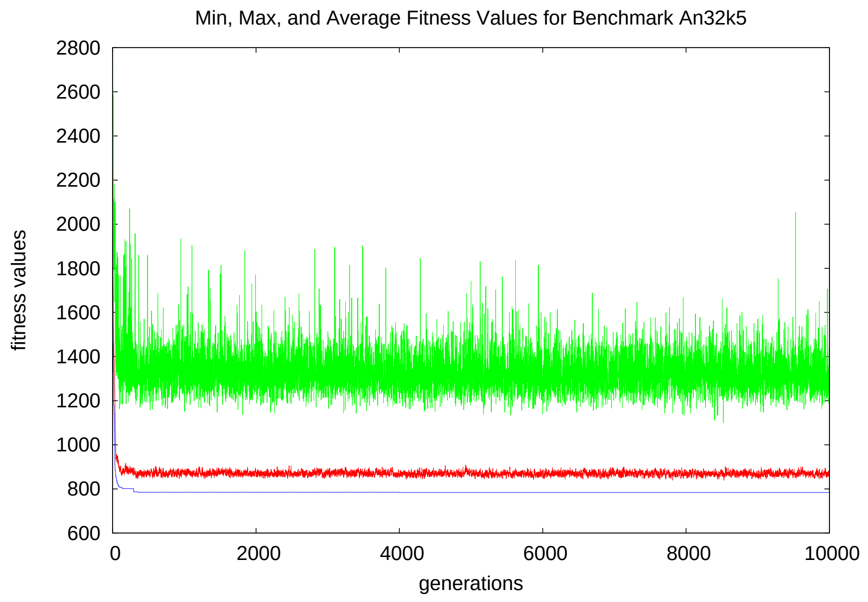
```
#!/sw/bin/gawk -f
##############################################################
#Matthew Russell 25 Oct 04
#
#Reads CVRP output files of the following format:
######################No blank header line
#X, Y, Demand
#X, Y, Demand
#...
#X, Y, Demand
#*TotalDemandForRoute*
#X, Y, Demand
#X, Y, Demand
#...
#X, Y, Demand
#####################No blank footer line
#Each route is separated by a delimeter *TotalDemandForRoute*
#This script creates the files necessary for gnuplot to create
#a nice visual of the solution in color with the nodes labelled
#according to their demand. Some text is also produced in the
#terminal that is useful for captioning. If you have trouble following
#the script, look at the output it produces and make sure you understand
#the basic control structure of how an awk script processed data.
#See the man page for more info.
##############################################################
BEGIN {
        num_routes=0
        curr_location=0
        print "set term postscript eps color solid 25" > ARGV[1] ".gnu"
        print "set nokey" >> ARGV[1] ".gnu"
        print "set size 1.5,1.5" >> ARGV[1] ".gnu"
        print "set pointsize 2.5" >> ARGV[1] ".gnu"
        print "set title \"" ARGV[1] "\"" >> ARGV[1] ".gnu"
        print "set out '/dev/null'" >> ARGV[1] ".gnu"

        print "\nDemand Information for " ARGV[1] ":"
}
{
        if(/\*([0-9]*)\*/) {
    print "\troute" num_routes " demand=" substr($1, 2, length($1)-2);
    num_routes += 1;
    }
else {
        print $1 " " $2 >> ARGV[1]"_route" num_routes
        x_coords[curr_location] = $1
        y_coords[curr_location] = $2
        demands[curr_location] = $3
        curr_location += 1
        }
}
END {
        plot_expression = "plot \"" ARGV[1] "_route0\" w lp pt 7 lw 3"
    for (i=0; i < num_routes; i++) {
                plot_expression = plot_expression ", \"" ARGV[1] "_route" i "\" w lp pt 7 lw 3"
        }
        print plot_expression >> ARGV[1] ".gnu"

        for (i=0; i < curr_location; i++) {
                print "set label \"" demands[i] "\" at first " x_coords[i] ", first "
                 (y_coords[i]+0.5) " font \"Times,14\" front center" >> ARGV[1] ".gnu"
        }

        print "set out '" ARGV[1] ".eps'" >> ARGV[1] ".gnu"
        print "replot" >> ARGV[1] ".gnu"
}
```

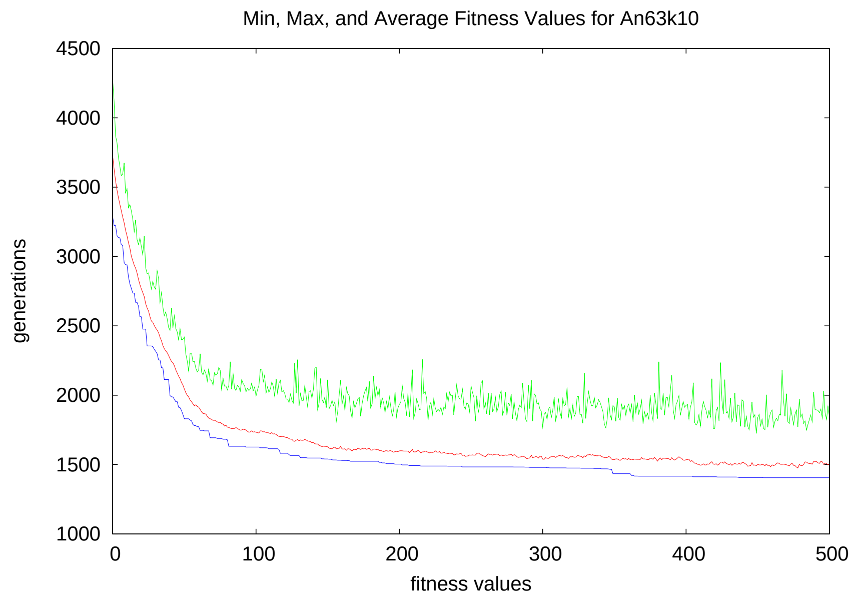
prob	pc	pm_swap	pm_displacement	pm_inversion	pm_insertion	popsize	num_gen	cap_pen	min_veh_pen	fitness
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	0	0	466
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	1	0	466
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	2	0	466
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	4	0	466
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	8	0	466
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	16	0	466
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	32	0	466
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	64	0	466
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	128	0	466
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	256	0	466
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	0	1	467
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	1	1	525.67
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	2	1	583.62
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	4	1	694.24
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	8	1	897.16
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	16	1	1200.2
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	32	1	1700.84
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	64	1	2207.88
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	128	1	2072
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	256	1	2057
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	0	2	468
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	1	2	584.62
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	2	2	695.24
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	4	2	795.16
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	8	2	910.84
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	16	2	1200.2
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	32	2	1685.68
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	64	2	2149.4
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	128	2	2069
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	256	2	2135
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	0	4	470
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	1	4	648.04
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	2	4	692.08
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	4	4	765.92
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	8	4	946.08
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	16	4	1200.2
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	32	4	1650.76
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	64	4	2124.44
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	128	4	2082
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	256	4	2079
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	0	8	474
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	1	8	653.07
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	2	8	692.08
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	4	8	765.92
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	8	8	910.84
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	16	8	1253.52

A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	32	8	1705.44
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	64	8	2087.96
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	128	8	2115.16
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	256	8	2040
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	0	16	482
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	1	16	657.87
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	2	16	715.4
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	4	16	765.92
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	8	16	910.84
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	16	16	1246.52
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	32	16	1621.92
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	64	16	2119.64
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	128	16	2138.16
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	256	16	2078
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	0	32	498
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	1	32	648.37
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	2	32	710.58
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	4	32	765.92
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	8	32	910.84
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	16	32	1200.2
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	32	32	1655.84
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	64	32	2113.96
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	128	32	2073
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	256	32	2080
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	0	64	530
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	1	64	648.37
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	2	64	692.08
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	4	64	765.92
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	8	64	910.84
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	16	64	1200.2
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	32	64	1655.16
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	64	64	2051.48
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	128	64	2132
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	256	64	2140
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	0	128	594
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	1	128	657.87
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	2	128	692.08
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	4	128	765.92
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	8	128	910.84
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	16	128	1200.2
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	32	128	1686.32
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	64	128	2113.96
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	128	128	2055
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	256	128	2062
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	0	256	598
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	1	256	648.04
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	2	256	692.08
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	4	256	765.92
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	8	256	910.84

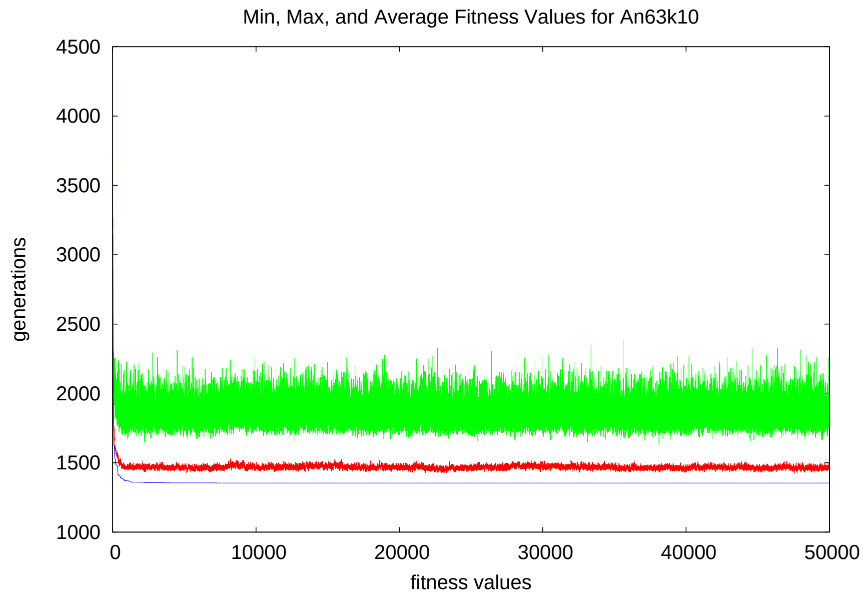
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	16	256	1200.2
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	32	256	1637.76
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	64	256	2163.68
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	128	256	2105
A-n32-k5	0.75	0.15	0.15	0.15	0.15	200	50000	256	256	2078


Min, Max, and Average Fitness Values for Benchmark An32k5
2800
2600
2400
2200
2000
1800
1600
1400
1200
1000
800
600
0
100
200
300
400
500
generations
fitness values

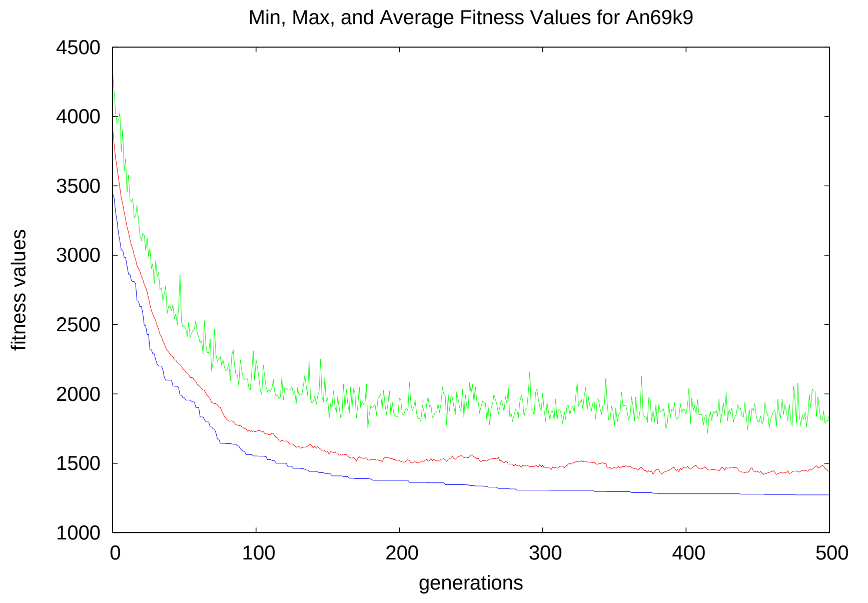

Min, Max, and Average Fitness Values for Benchmark An32k5
2800
2600
2400
2200
2000
1800
1600
1400
1200
1000
800
600
0
2000
4000
6000
8000
10000
generations
fitness values

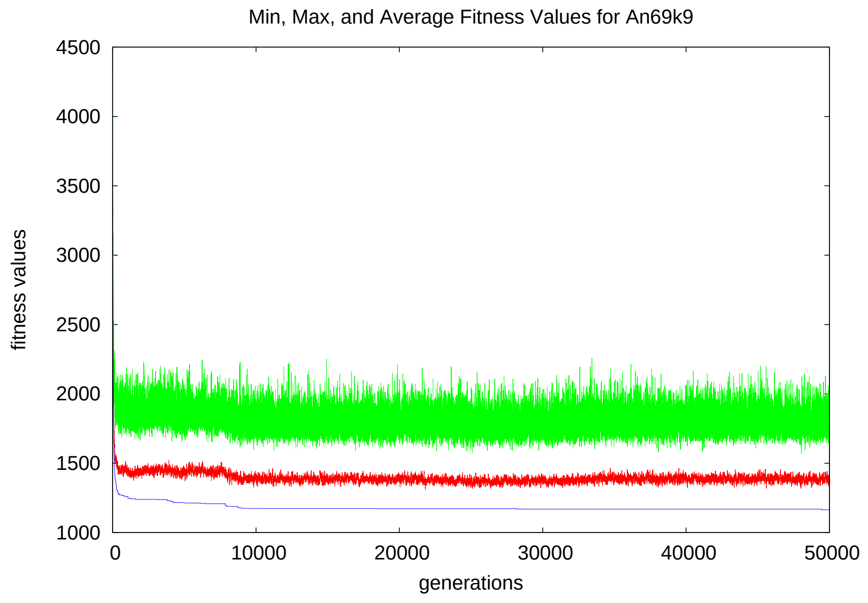


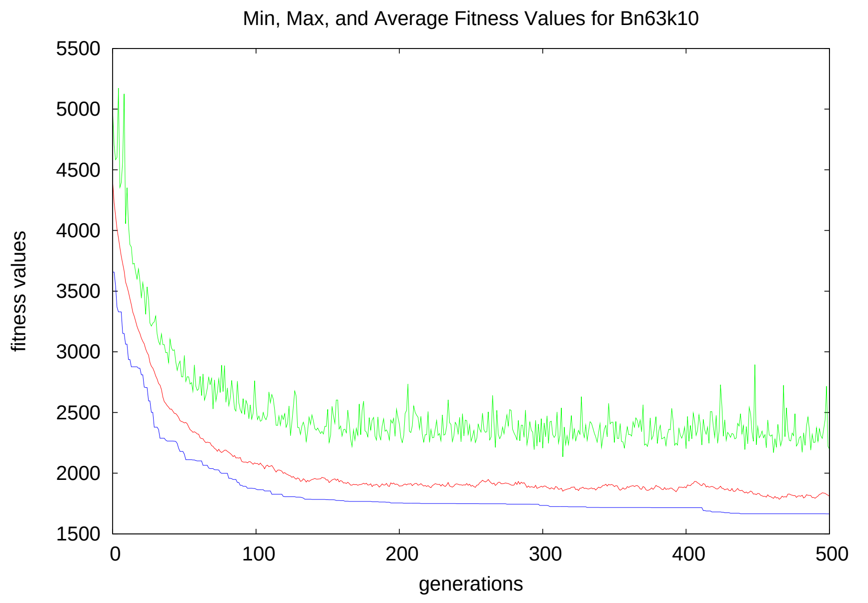
Min, Max, and Average Fitness Values for An63k10
generations
fitness values
4500
4000
3500
3000
2500
2000
1500
1000
0
100
200
300
400
500

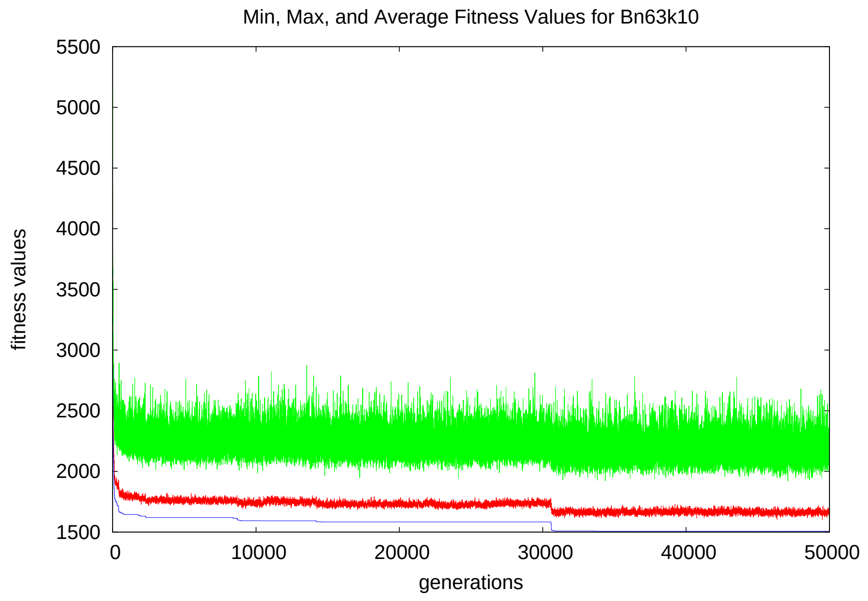


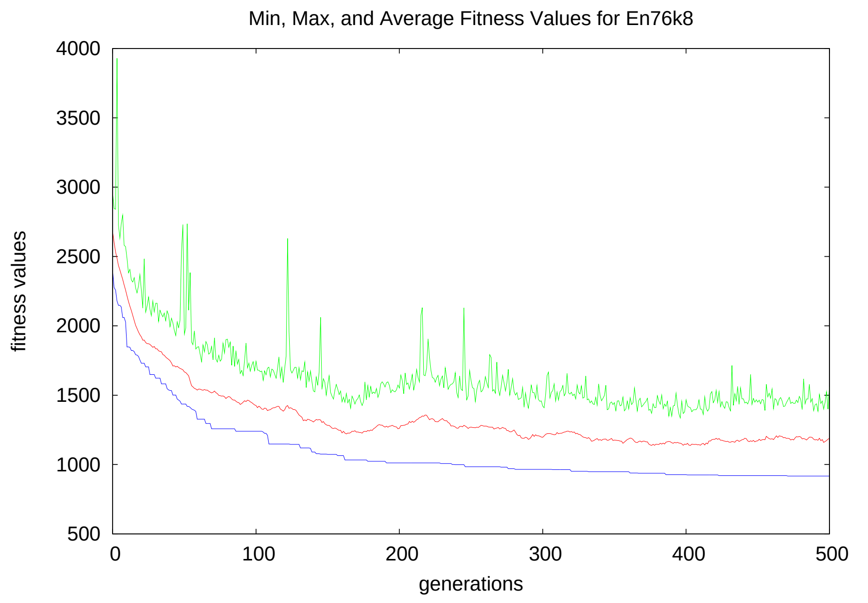
Min, Max, and Average Fitness Values for An63k10
generations
fitness values
4500
4000
3500
3000
2500
2000
1500
1000
0
10000
20000
30000
40000
50000


Min, Max, and Average Fitness Values for An69k9
fitness values
4500
4000
3500
3000
2500
2000
1500
1000
0
100
200
300
400
500
generations

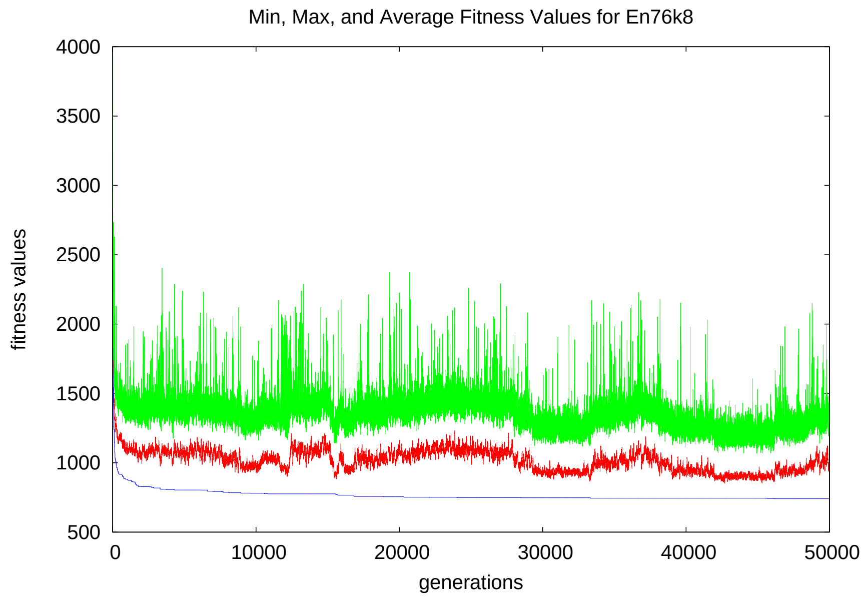

Min, Max, and Average Fitness Values for An69k9
fitness values
4500
4000
3500
3000
2500
2000
1500
1000
0
10000
20000
30000
40000
50000
generations


Min, Max, and Average Fitness Values for Bn63k10
fitness values
5500
5000
4500
4000
3500
3000
2500
2000
1500
0
100
200
300
400
500
generations

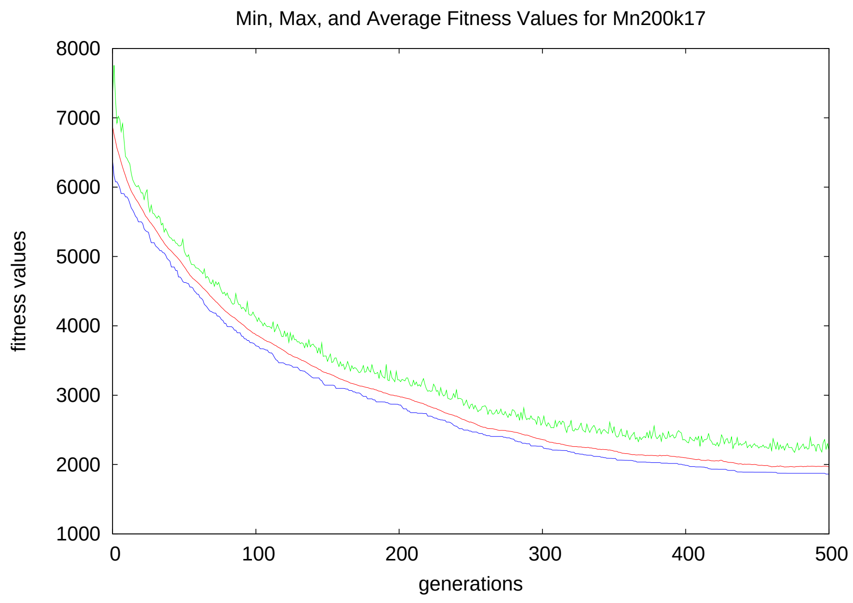

Min, Max, and Average Fitness Values for Bn63k10
fitness values
5500
5000
4500
4000
3500
3000
2500
2000
1500
0
10000
20000
30000
40000
50000
generations

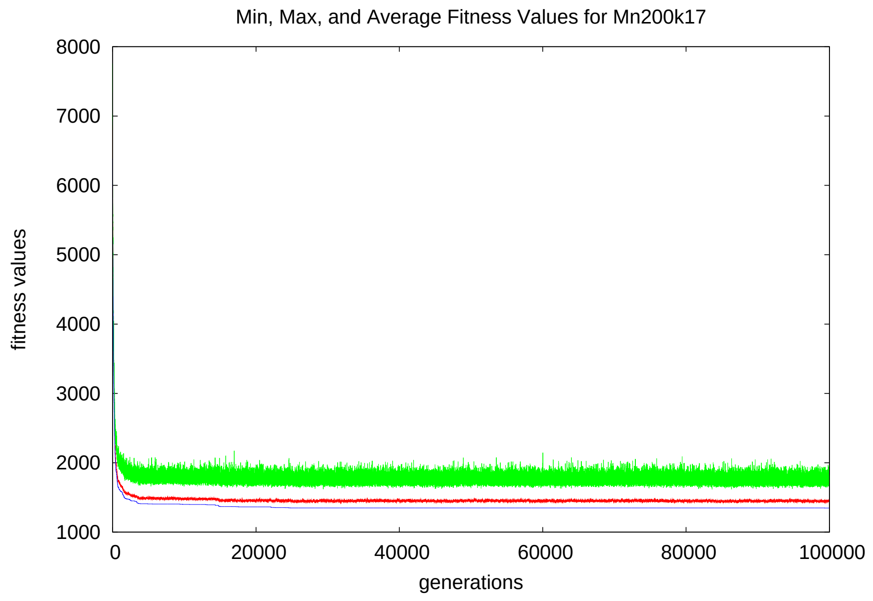


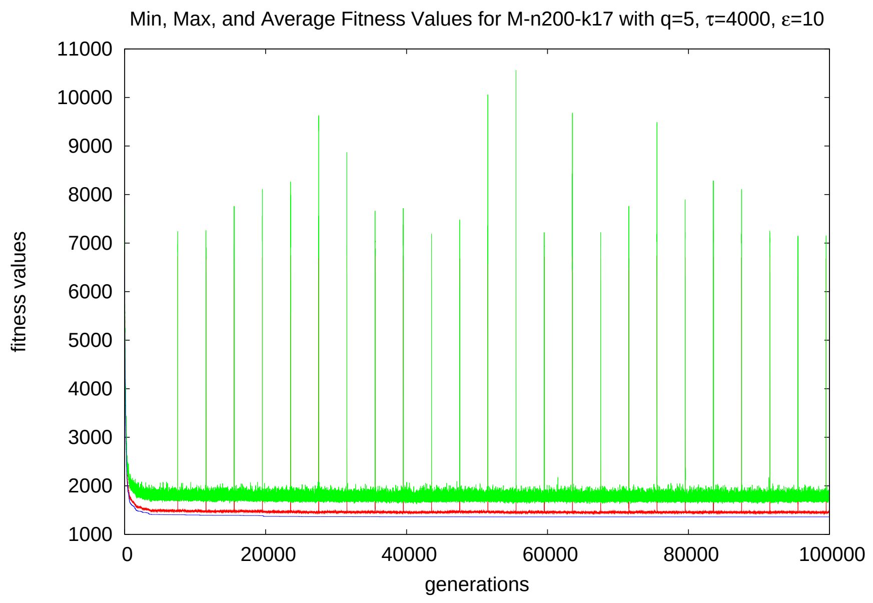
Min, Max, and Average Fitness Values for En76k8
fitness values
generations
4000
3500
3000
2500
2000
1500
1000
500
0
100
200
300
400
500

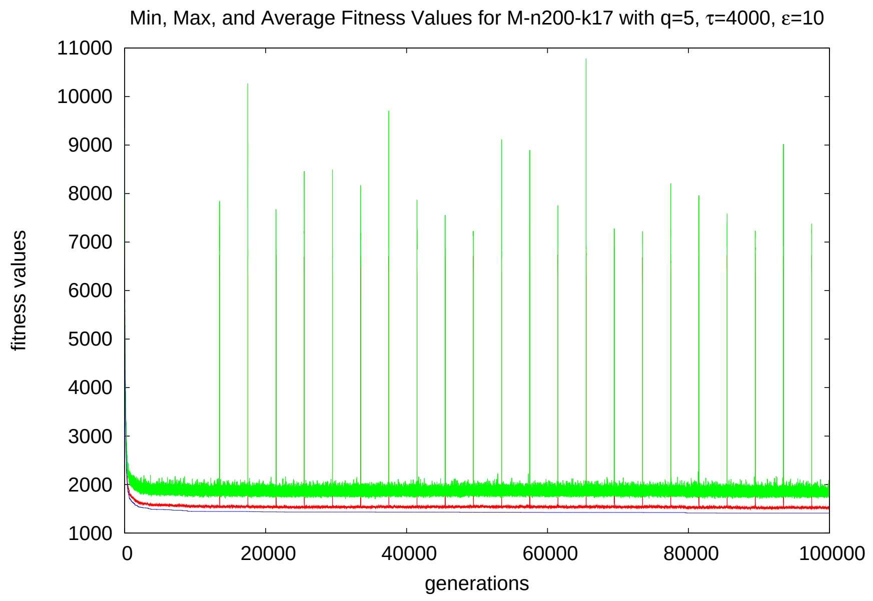


Min, Max, and Average Fitness Values for En76k8
fitness values
generations
4000
3500
3000
2500
2000
1500
1000
500
0
10000
20000
30000
40000
50000

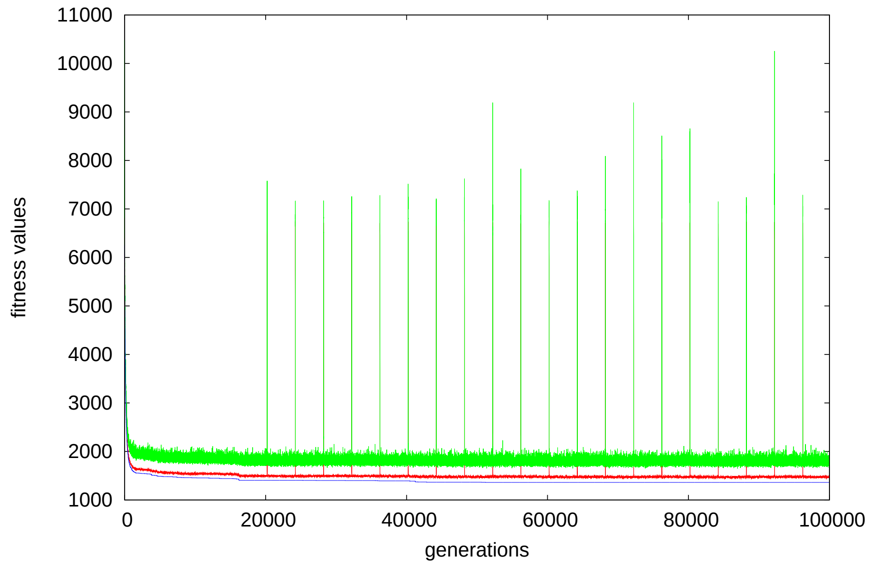

Min, Max, and Average Fitness Values for Mn200k17
fitness values
8000
7000
6000
5000
4000
3000
2000
1000
0
100
200
300
400
500
generations


Min, Max, and Average Fitness Values for Mn200k17
fitness values
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000
generations

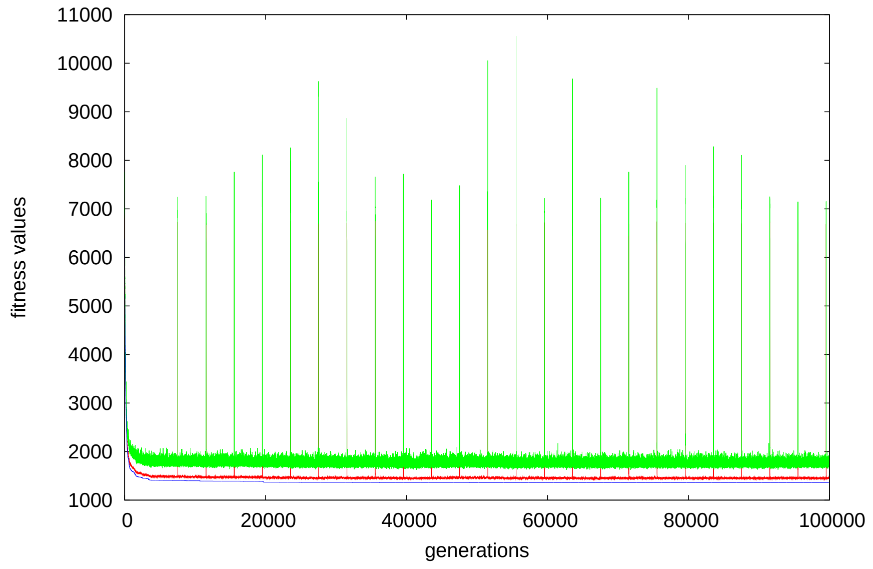

Min, Max, and Average Fitness Values for M-n200-k17 with q=5, τ=4000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000


Min, Max, and Average Fitness Values for M-n200-k17 with q=5, τ=4000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

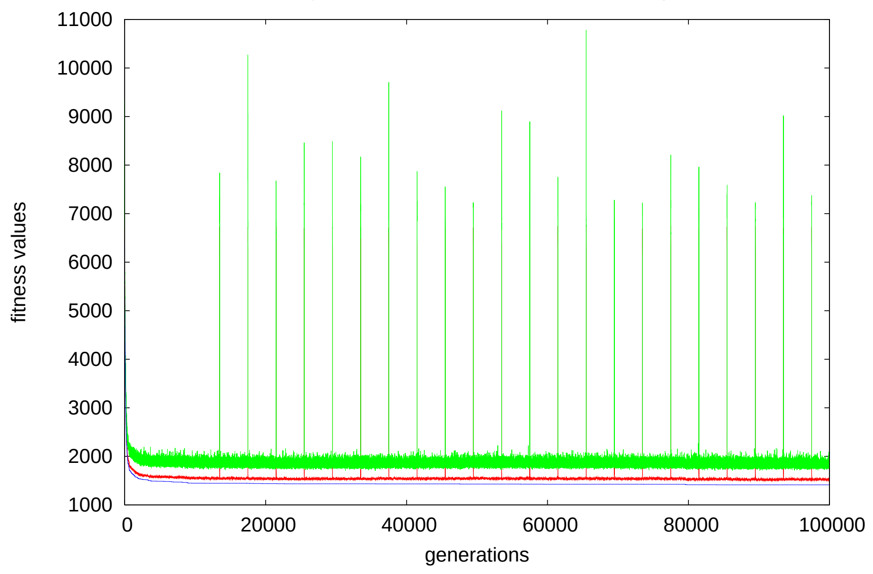
Min, Max, and Average Fitness Values for M-n200-k17 with q=5, τ=4000, ε=10



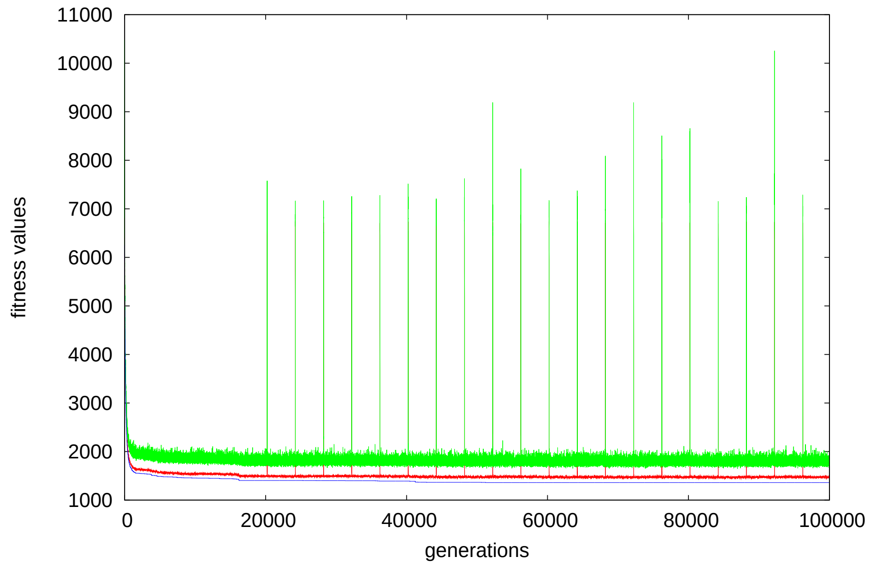
Min, Max, and Average Fitness Values for M-n200-k17 with q=10, τ=4000, ε=10

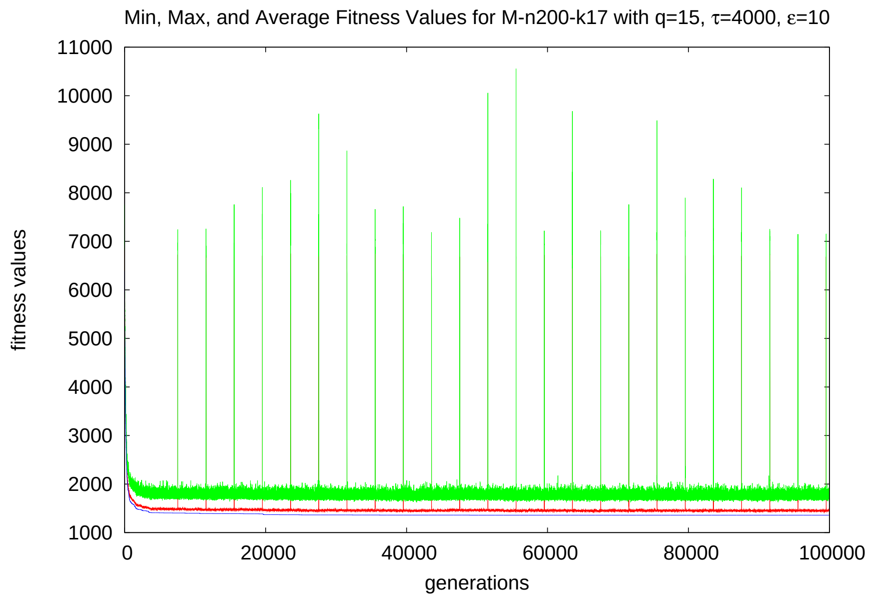


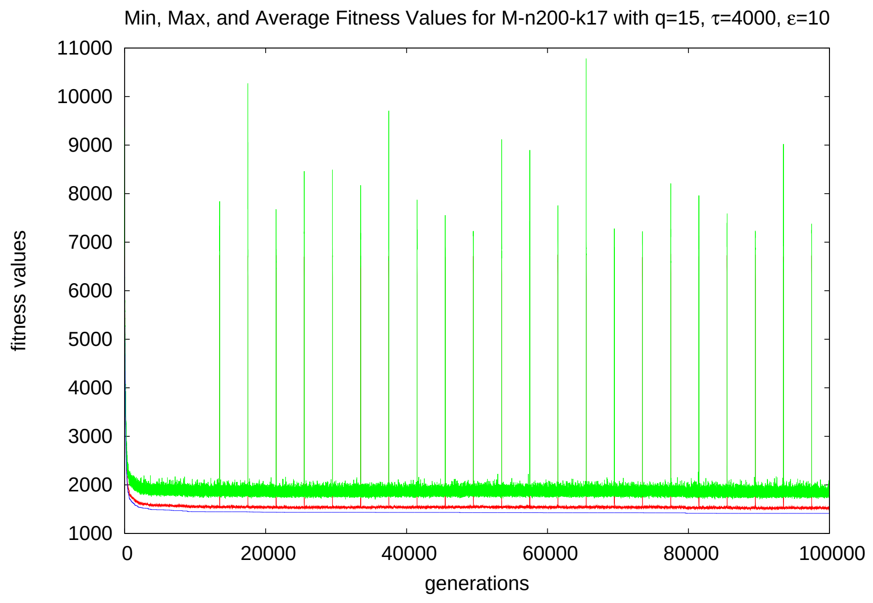
Min, Max, and Average Fitness Values for M-n200-k17 with q=10, τ=4000, ε=10

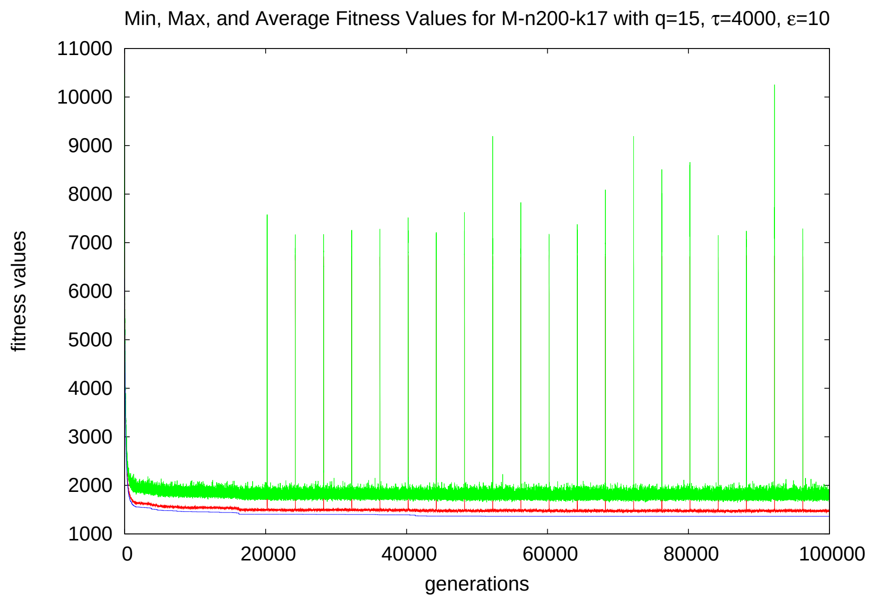


Min, Max, and Average Fitness Values for M-n200-k17 with q=10, τ=4000, ε=10

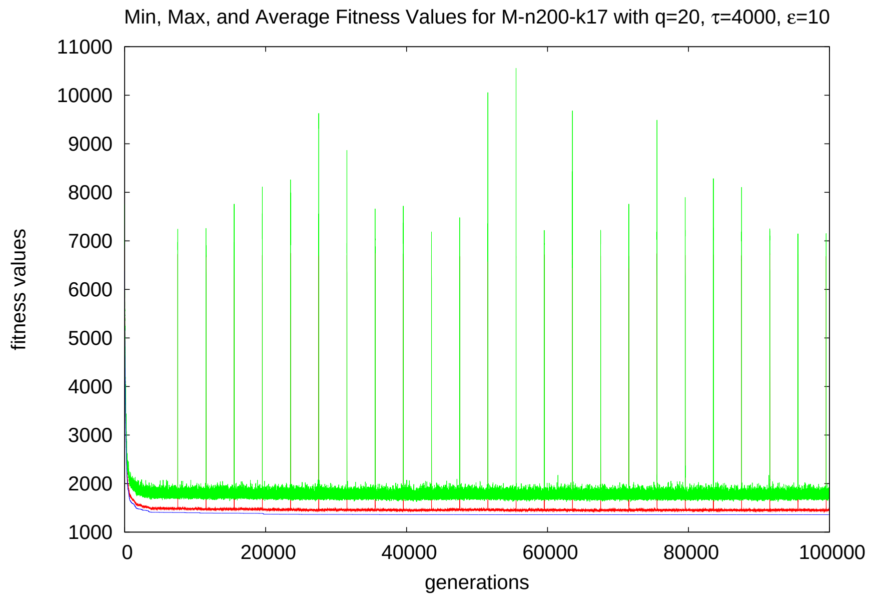



Min, Max, and Average Fitness Values for M-n200-k17 with q=15, τ=4000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

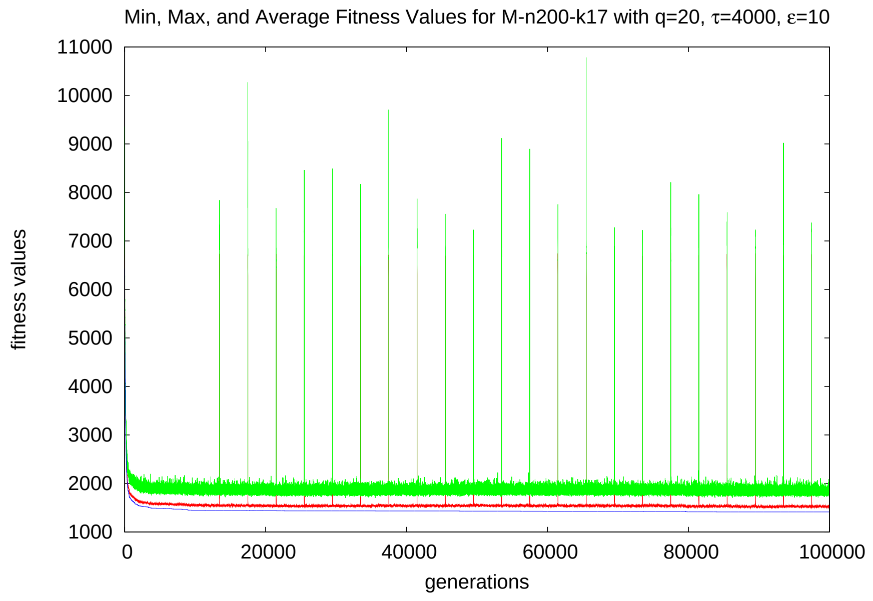

Min, Max, and Average Fitness Values for M-n200-k17 with q=15, τ=4000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

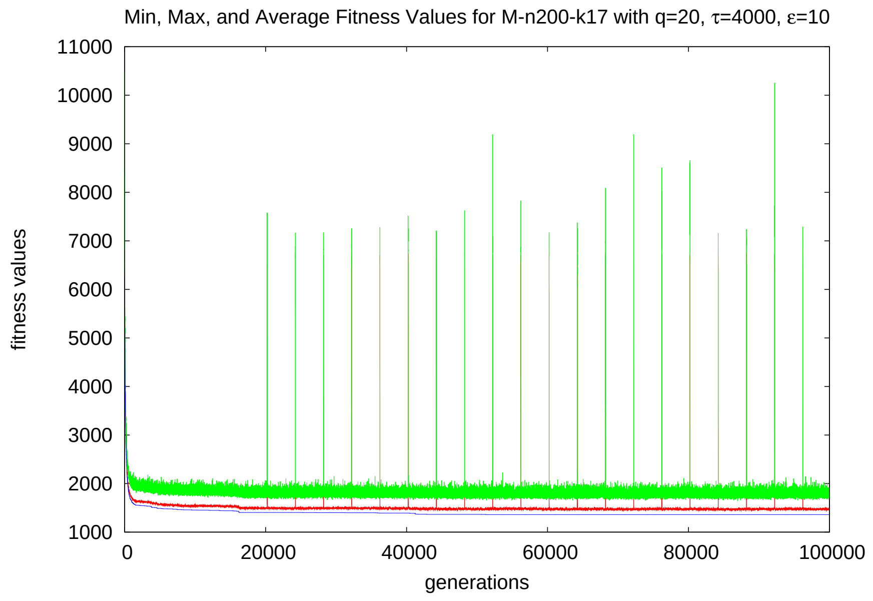


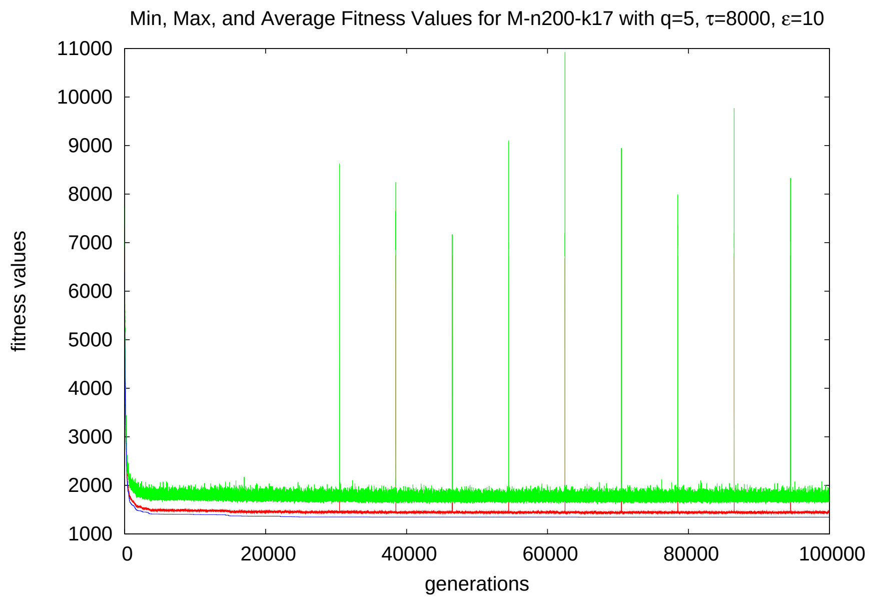
Min, Max, and Average Fitness Values for M-n200-k17 with q=15, τ=4000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

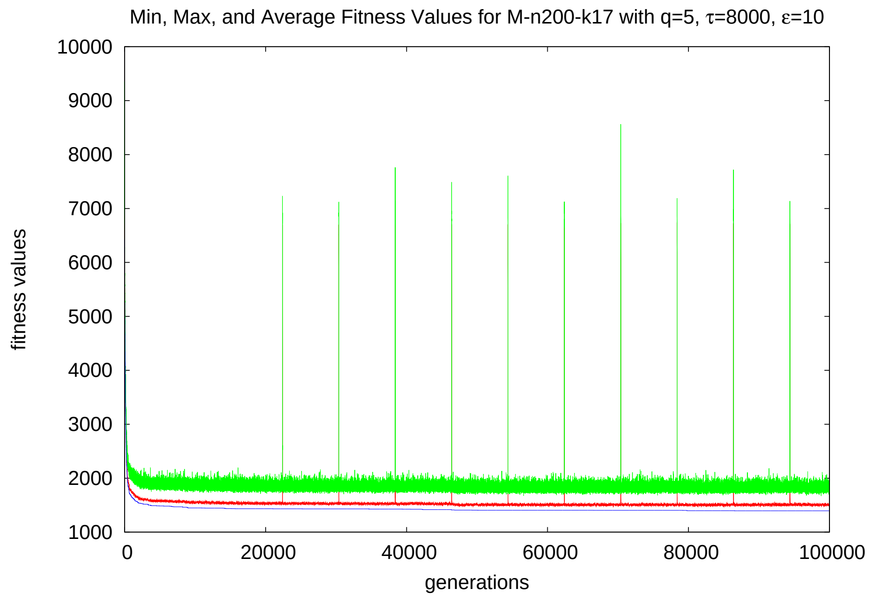


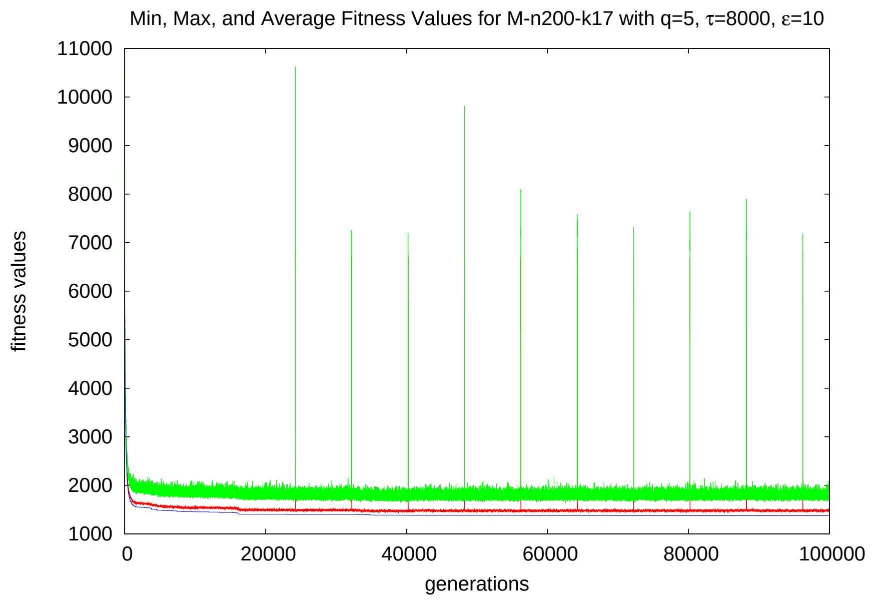
Min, Max, and Average Fitness Values for M-n200-k17 with q=20, τ=4000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

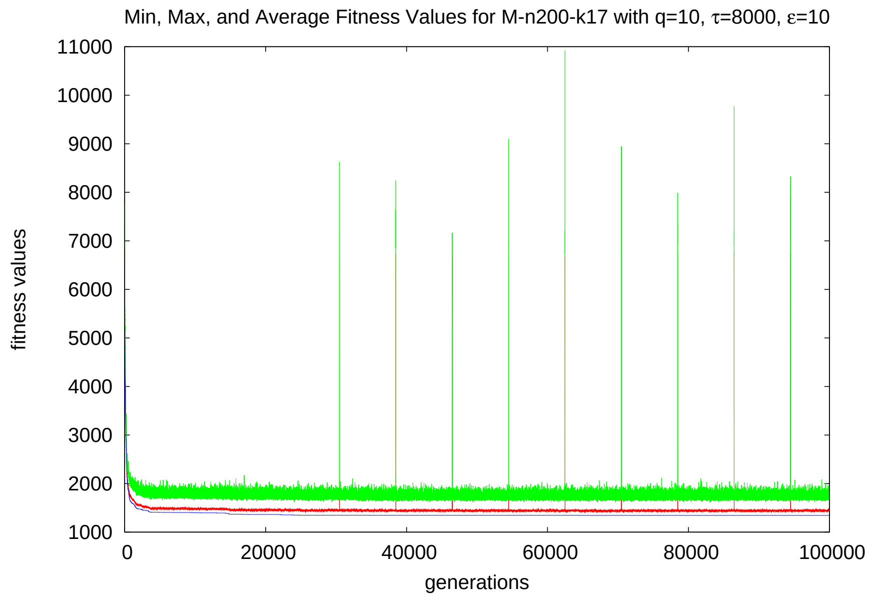

Min, Max, and Average Fitness Values for M-n200-k17 with q=20, τ=4000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

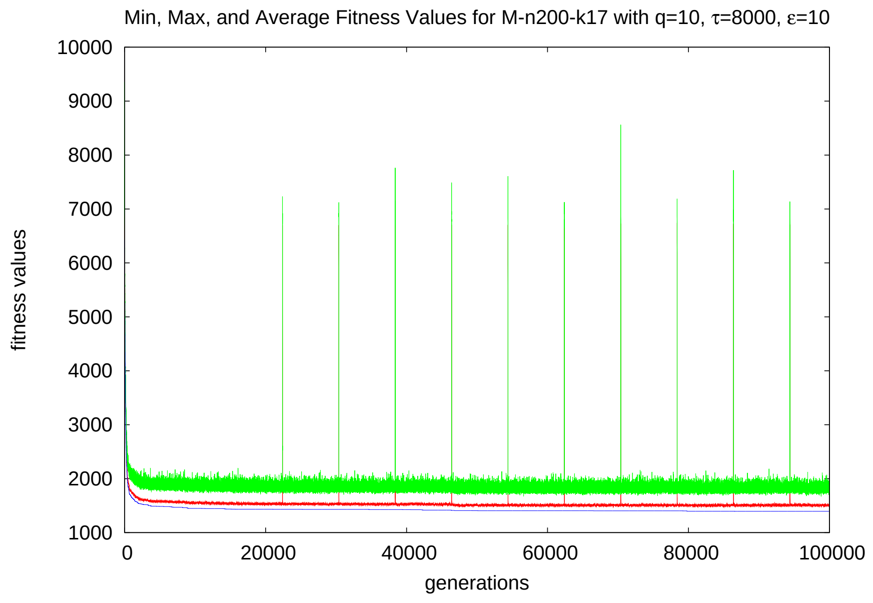

Min, Max, and Average Fitness Values for M-n200-k17 with q=20, τ=4000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

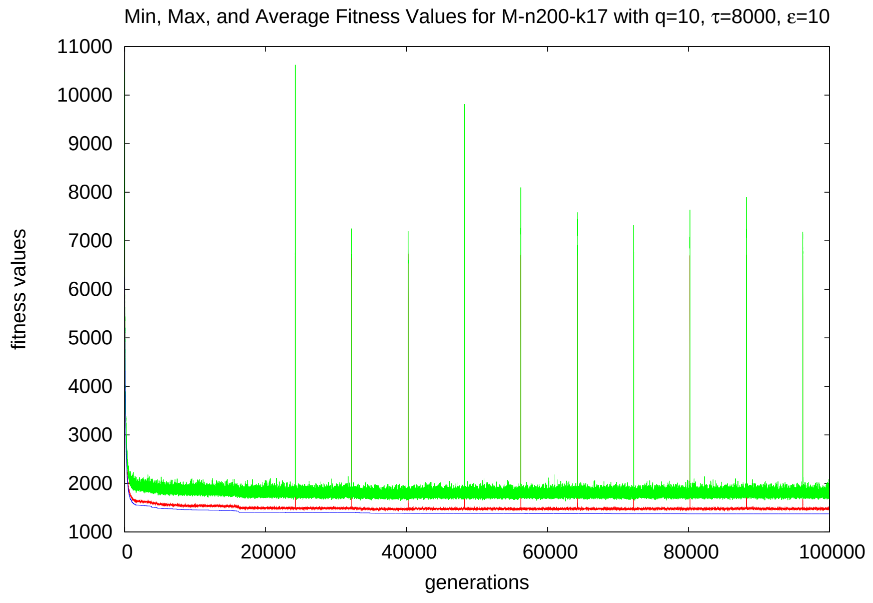

Min, Max, and Average Fitness Values for M-n200-k17 with q=5, τ=8000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

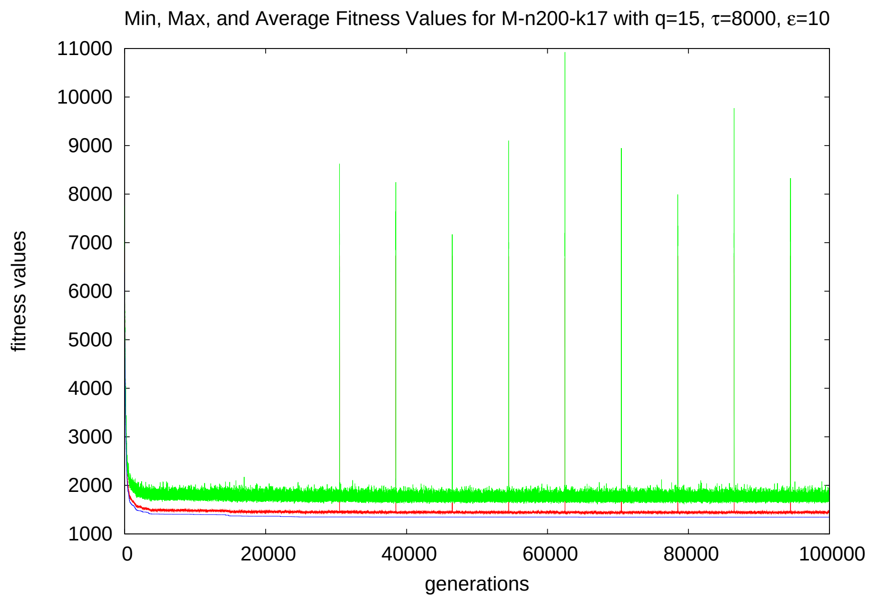

Min, Max, and Average Fitness Values for M-n200-k17 with q=5, τ=8000, ε=10
fitness values
generations
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

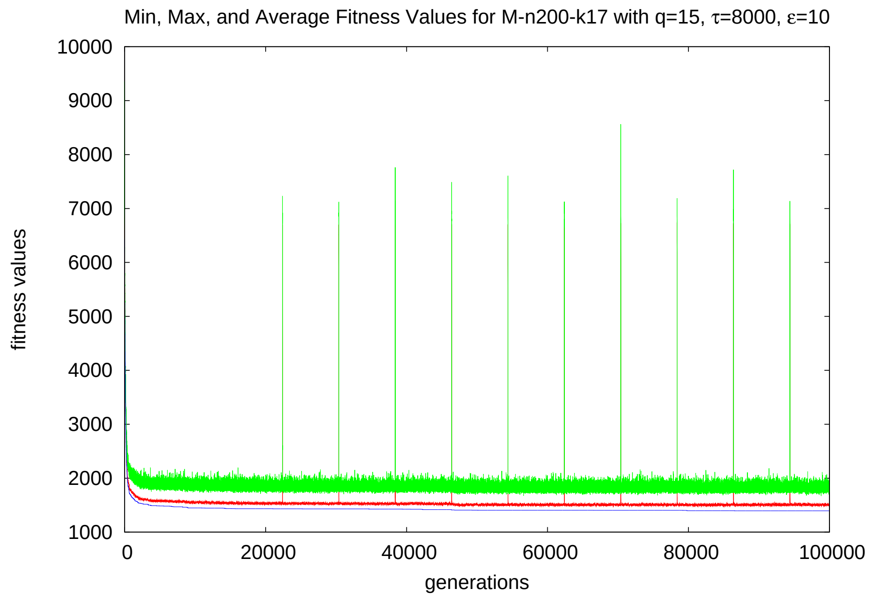

Min, Max, and Average Fitness Values for M-n200-k17 with q=5, τ=8000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

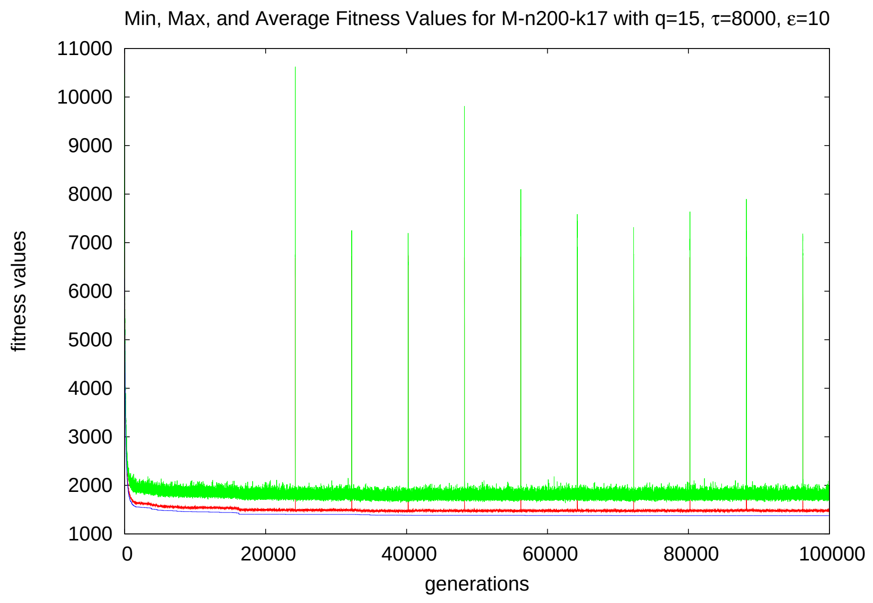

Min, Max, and Average Fitness Values for M-n200-k17 with q=10, τ=8000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000


Min, Max, and Average Fitness Values for M-n200-k17 with q=10, τ=8000, ε=10
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
fitness values
0
20000
40000
60000
80000
100000
generations

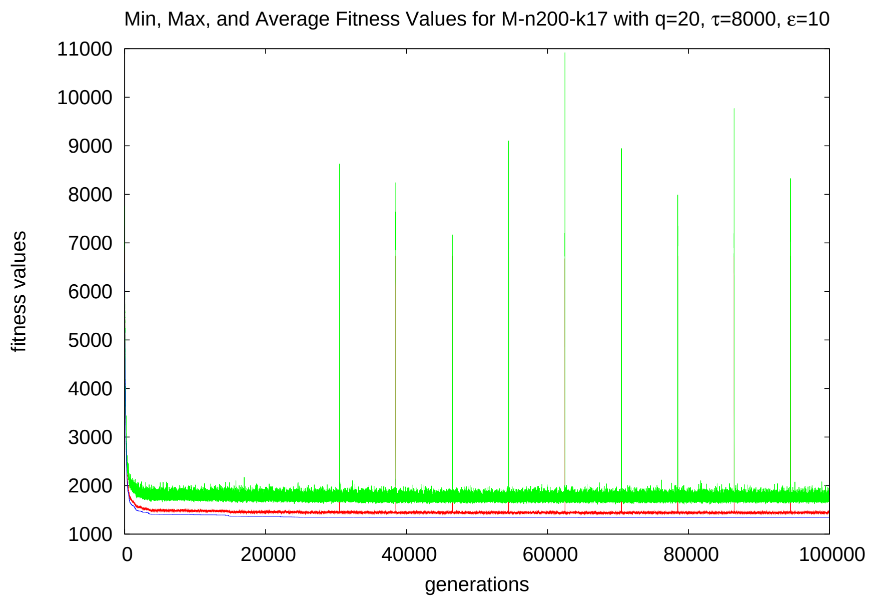

Min, Max, and Average Fitness Values for M-n200-k17 with q=10, τ=8000, ε=10
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
fitness values
0
20000
40000
60000
80000
100000
generations


Min, Max, and Average Fitness Values for M-n200-k17 with q=15, τ=8000, ε=10
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
fitness values
0
20000
40000
60000
80000
100000
generations

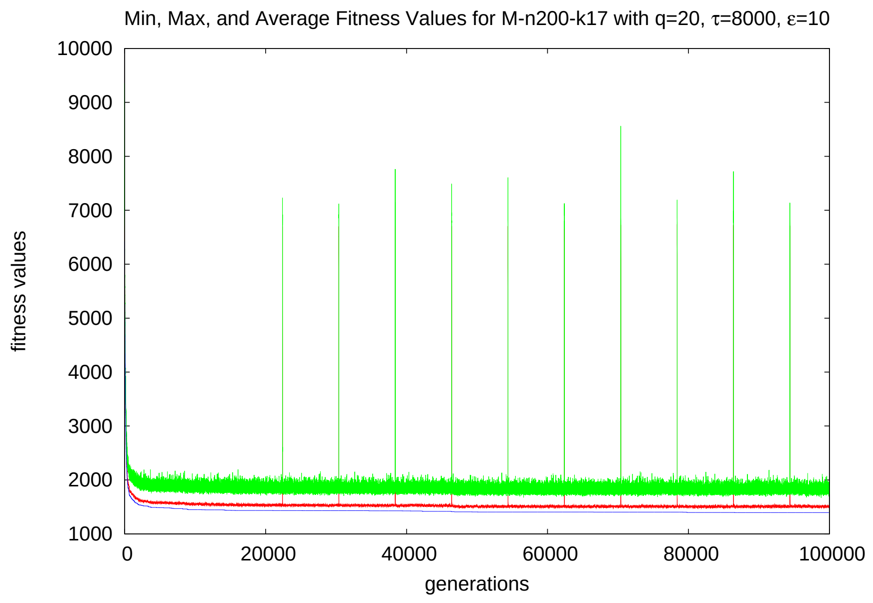

Min, Max, and Average Fitness Values for M-n200-k17 with q=15, τ=8000, ε=10
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
fitness values
0
20000
40000
60000
80000
100000
generations

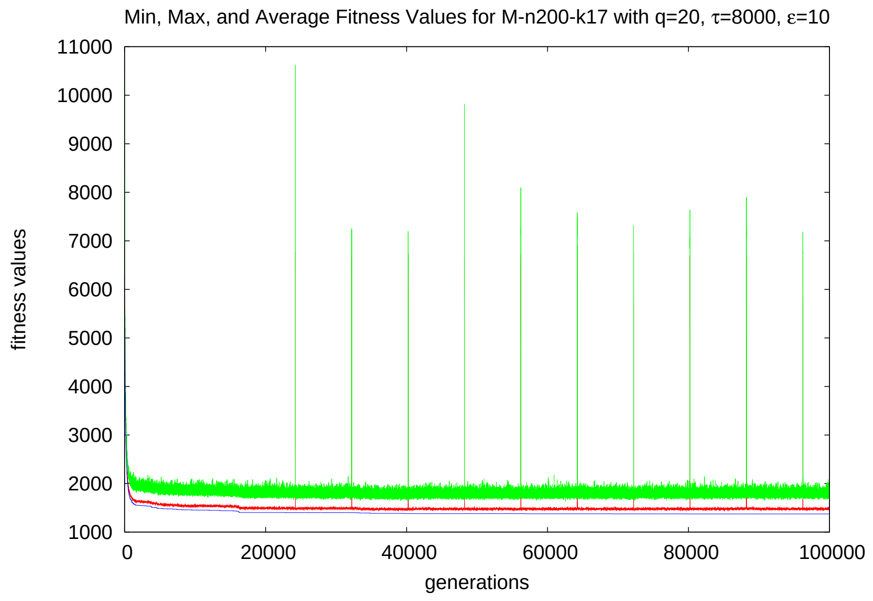


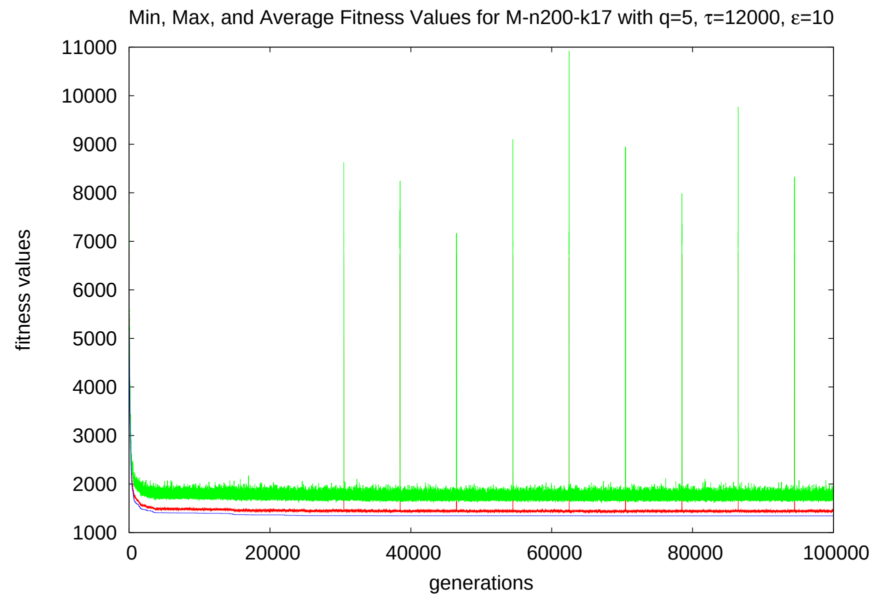
Min, Max, and Average Fitness Values for M-n200-k17 with q=15, τ=8000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

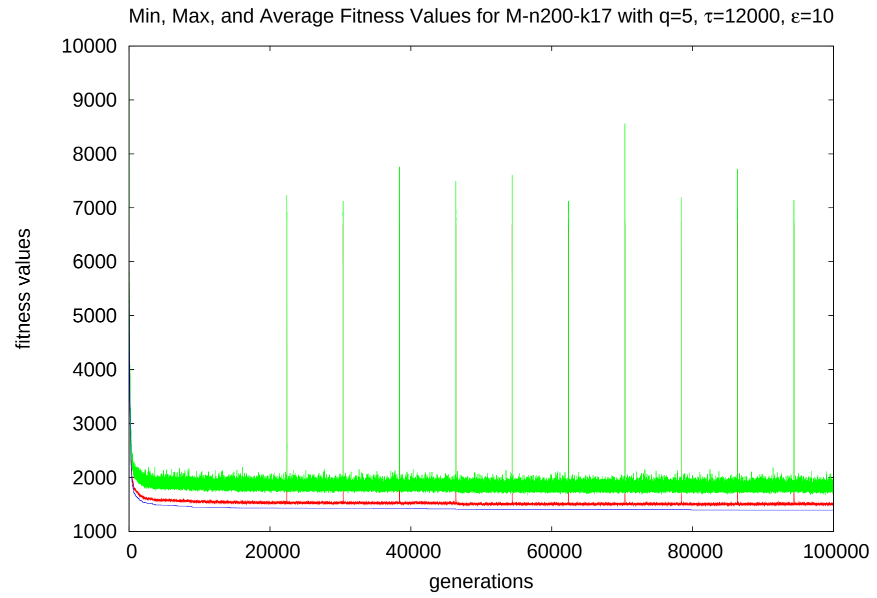


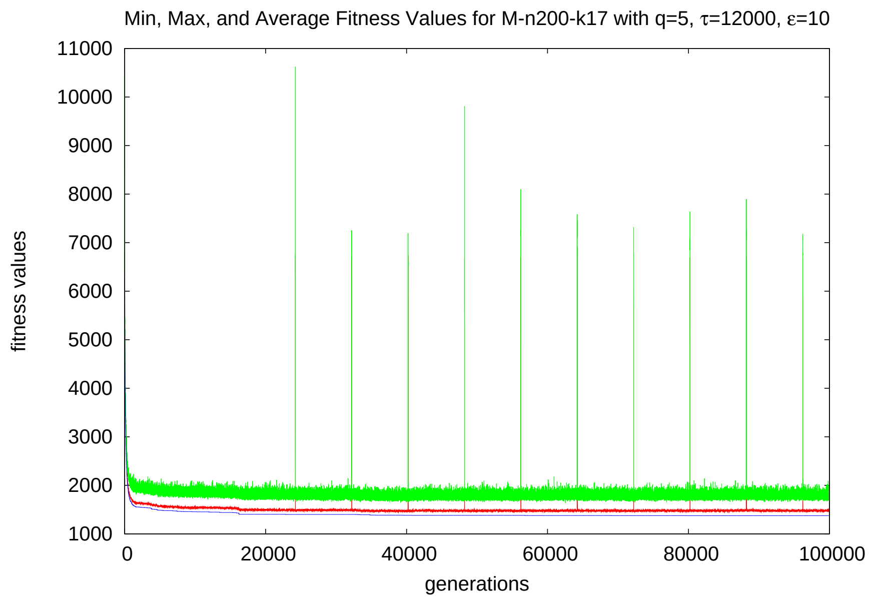
Min, Max, and Average Fitness Values for M-n200-k17 with q=20, τ=8000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000


Min, Max, and Average Fitness Values for M-n200-k17 with q=20, τ=8000, ε=10
fitness values
generations
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

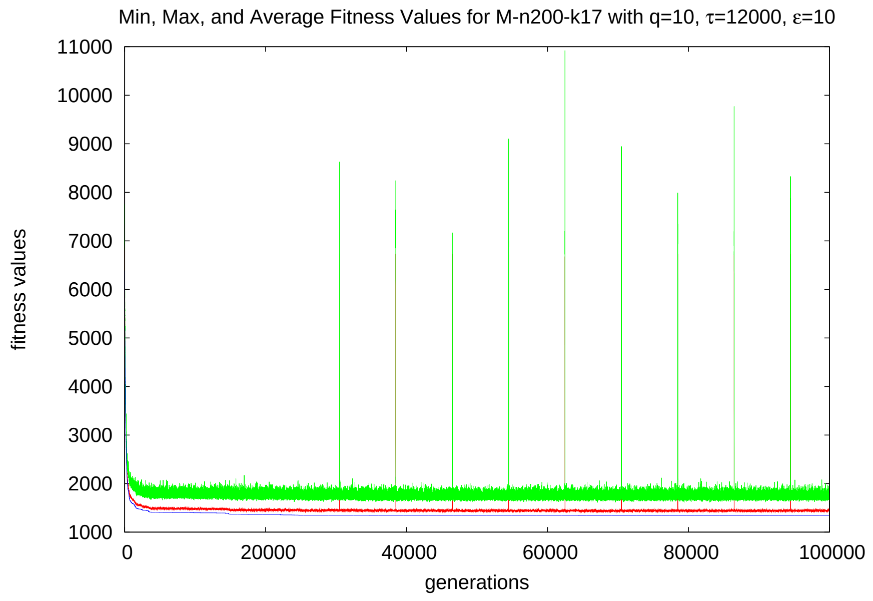

Min, Max, and Average Fitness Values for M-n200-k17 with q=20, τ=8000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000


Min, Max, and Average Fitness Values for M-n200-k17 with q=5, τ=12000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

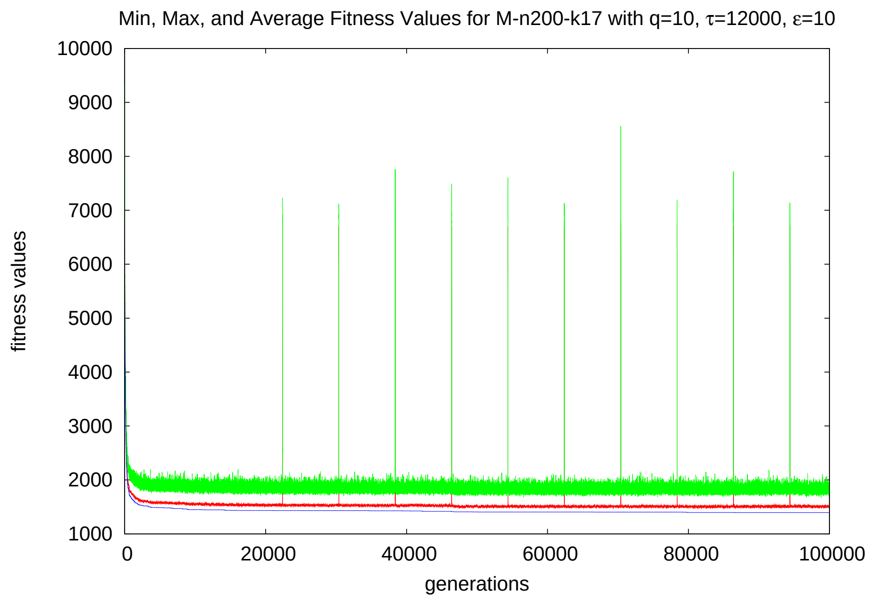

Min, Max, and Average Fitness Values for M-n200-k17 with q=5, τ=12000, ε=10
fitness values
generations
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

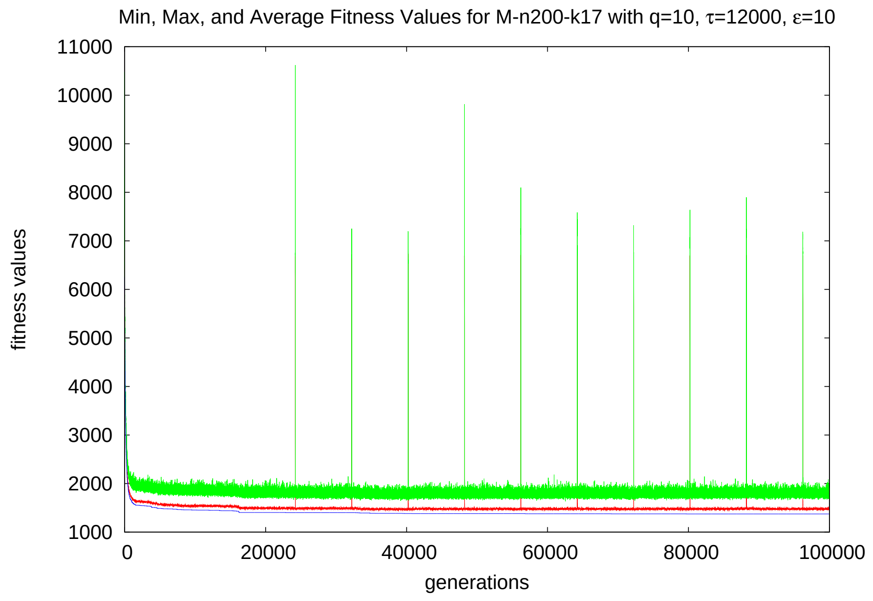


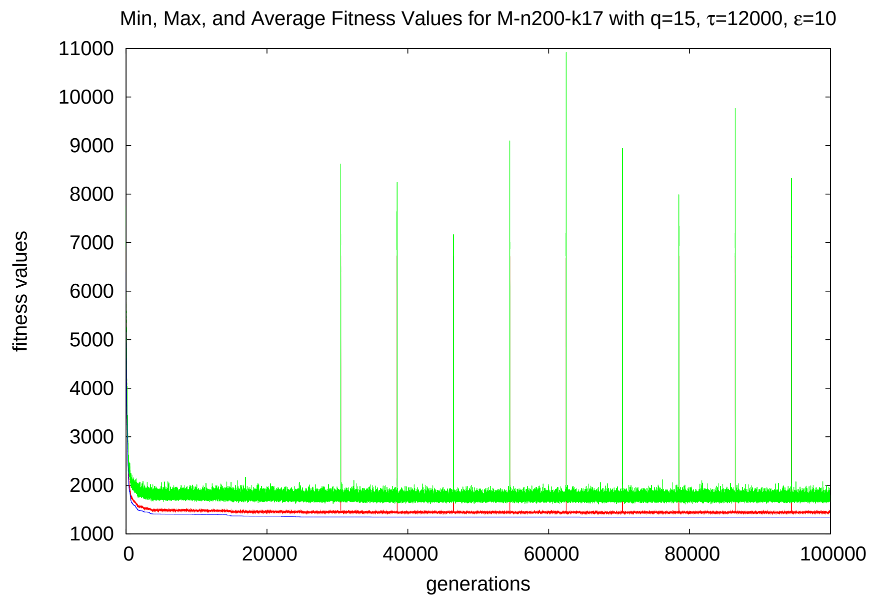
Min, Max, and Average Fitness Values for M-n200-k17 with q=5, τ=12000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

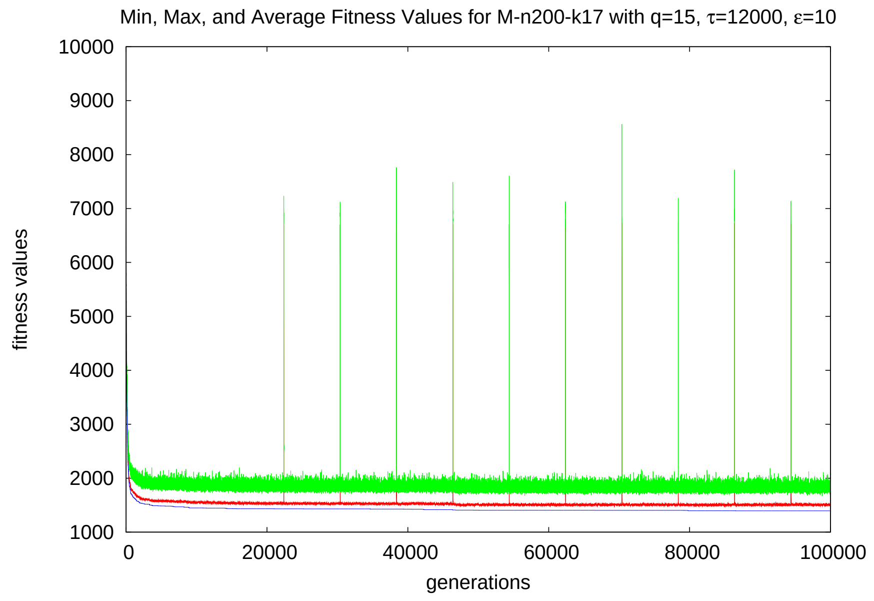


Min, Max, and Average Fitness Values for M-n200-k17 with q=10, τ=12000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

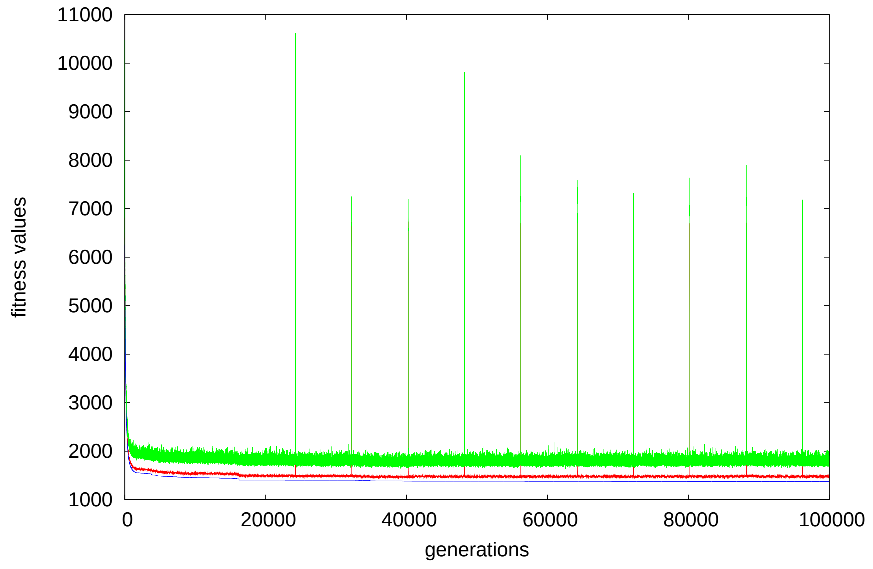

Min, Max, and Average Fitness Values for M-n200-k17 with q=10, τ=12000, ε=10
fitness values
generations
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000


Min, Max, and Average Fitness Values for M-n200-k17 with q=10, τ=12000, ε=10
fitness values
generations
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000

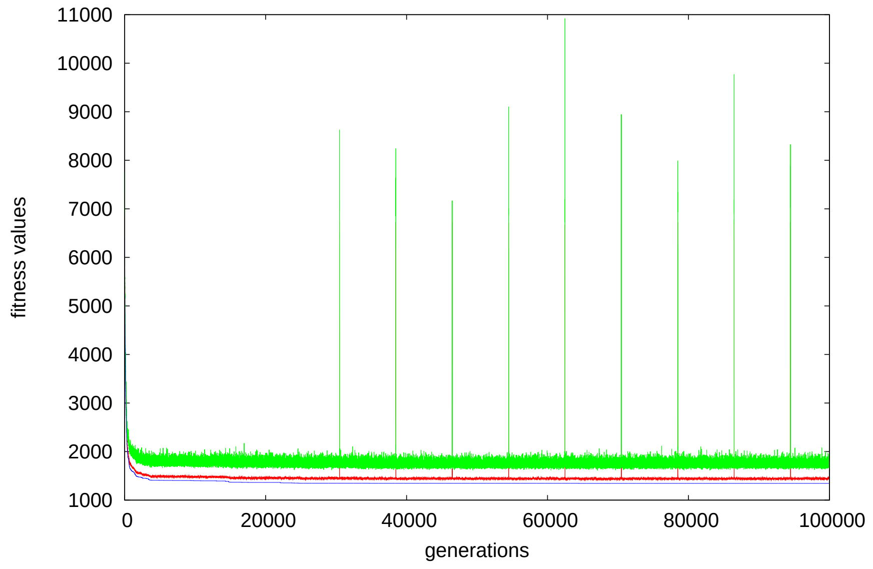

Min, Max, and Average Fitness Values for M-n200-k17 with q=15, τ=12000, ε=10
fitness values
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000
generations

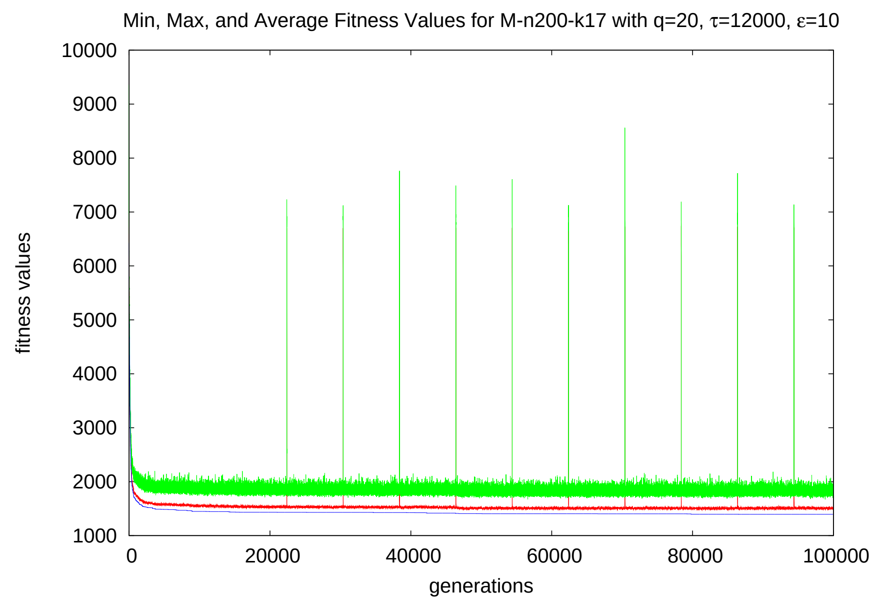

Min, Max, and Average Fitness Values for M-n200-k17 with q=15, τ=12000, ε=10
fitness values
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000
generations

Min, Max, and Average Fitness Values for M-n200-k17 with q=15, τ=12000, ε=10

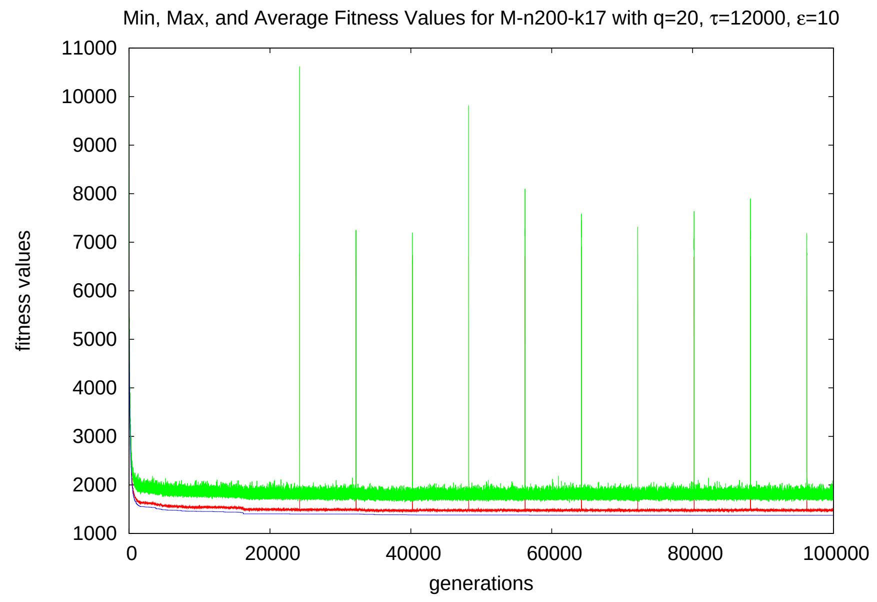


Min, Max, and Average Fitness Values for M-n200-k17 with q=20, τ=12000, ε=10





Min, Max, and Average Fitness Values for M-n200-k17 with q=20, τ=12000, ε=10
fitness values
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000
generations



Min, Max, and Average Fitness Values for M-n200-k17 with q=20, τ=12000, ε=10
fitness values
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
20000
40000
60000
80000
100000
generations

Appendix I. Technical Notes on Compiling GALib and Auxiliary Code

I.1 Compilation Notes

GALib comes with excellent compilation instructions and should be a standard compile for 32-bit and 64-bit Linux system alike, although some warnings may appear and continue to appear as the GCC compiler becomes more strict. The end result of the compile is a static library that must be recompiled each time any code changes to source code the library uses to build. Such changes include explicit source code changes, but also changes to preprocessor statements if a different random number generator is to be used.

I.2 Auxiliary Code

The following three code snippets are necessary for compiling the C++ project containing source file GVRGenome.h, which depends on the RandomTournamentSelector class and produces a single offspring from its crossover operators.

```
/* ---------------------------------------------------------------------------
RandomTournamentSelector


GALib surprisingly did not come with a tournament selection operator that chooses members from the t
the prototype for the function definition.
--------------------------------------------------------------------------- */
class GARandomTournamentSelector : public GASelectionScheme {
public:
  GADefineIdentity("GARandomTournamentSelector", 123456);


  GARandomTournamentSelector(int q=2, int w=GASelectionScheme::RAW) : GASelectionScheme(w) {tourney_
  GARandomTournamentSelector(const GARandomTournamentSelector& orig) { copy(orig); }
  GARandomTournamentSelector& operator=(const GASelectionScheme& orig)
    { if(&orig != this) copy(orig); return *this; }
  virtual ~GARandomTournamentSelector() { }
  virtual GASelectionScheme* clone() const { return new GARandomTournamentSelector; }
  virtual GAGenome& select() const;

private:
int tourney_size;
};



/* ----------------------------------------------------------------------------
RandomTournamentSelector

GALib surprisingly did not come with a tournament selection operator that chooses members from the t
---------------------------------------------------------------------------- */
// Randomly select q individuals from the population.  This selector does not
// care whether it operates on the fitness or objective scores. If a minimization
//problem, it returns the lowest score. if a maximization problem, it returns
//the highest
GAGenome&
GARandomTournamentSelector::select() const {
    float best_score = 0.0;
    int individual;
    int best_individual;
    bool maximize = true;
```

```
//determine if this is a minimization or maximization problem
if(pop->order() == GAPopulation::LOW_IS_BEST) {
    maximize = false;
    best_score = MAXFLOAT;
 }


for (int i=0; i < tourney_size; i++) {
    individual = GARandomInt(0, pop->size()-1);

    if (maximize) {
        if (pop->individual(individual).score() > best_score) {
            best_individual = individual;
            best_score = pop->individual(individual).score();
        }
    }
    else {
        if (pop->individual(individual).score() < best_score) {
            best_individual = individual;
            best_score = pop->individual(individual).score();
        }
    }

}
```

```
  return pop->individual(best_individual);

}


/* -------------------------------------------------------------------------

Changes to GASimpleGA::step() contained in GASelector.C


These changes were necessary because GVRGenome.h

produces one offspring (not two) from a crossover. The

GALib routines appear to all implicity expect two offspring.

------------------------------------------------------------------------- */


void

GASimpleGA::step()

{

  int i, mut, c1, c2;

  GAGenome *mom, *dad;          // tmp holders for selected genomes


  GAPopulation *tmppop;     // Swap the old population with the new pop.

  tmppop = oldPop;      // When we finish the ++ we want the newly

  oldPop = pop;         // generated population to be current (for

  pop = tmppop;         // references to it from member functions).


// Generate the individuals in the temporary population from individuals in

// the main population.


  for(i=0; i<pop->size()-1; i+=2){  // takes care of odd population
```

```
 mom = &(oldPop->select());
 dad = &(oldPop->select());
 stats.numsel += 2;      // keep track of number of selections


 c1 = c2 = 0;
 if(GAFlipCoin(pCrossover())){
int num_children =0;
   stats.numcro += num_children = (*scross)(*mom, *dad,
             &pop->individual(i), &pop->individual(i+1));
 c1 = 1;
   //Matthew Russell 10 Oct 04: if doing a single child crossover,
   //pop->individual(i+1) needs to not be null, so go ahead and do another
   //crossover for that one
   //This isn't exactly the cleanest way to do it (example 22 is arguably cleaner)
   //but this generalizes
   //a solution for now.
   if (GAFlipCoin(pCrossover())) {
     if (num_children ==1) {
         mom = &(oldPop->select());
         dad = &(oldPop->select());
         stats.numsel += 2;      // keep track of number of selections


     stats.numcro += (*scross)(*mom, *dad,
             &pop->individual(i+1), 0);
     c1 = c2 = 1;
```

```
       }//if

     }//if

     else

         pop->individual(i+1).copy(*mom);

   }

else{

     pop->individual( i ).copy(*mom);

     pop->individual(i+1).copy(*dad);

   }

   stats.nummut += (mut = pop->individual( i ).mutate(pMutation()));

   if(mut > 0) c1 = 1;

   stats.nummut += (mut = pop->individual(i+1).mutate(pMutation()));

   if(mut > 0) c2 = 1;


   stats.numeval += c1 + c2;

 }

 if(pop->size() % 2 != 0){ // do the remaining population member

   mom = &(oldPop->select());

   dad = &(oldPop->select());

   stats.numsel += 2;      // keep track of number of selections


   c1 = 0;

   if(GAFlipCoin(pCrossover())){

     stats.numcro += (*scross)(*mom, *dad, &pop->individual(i), (GAGenome*)0);

     c1 = 1;

   }
```

```
    else{
      if(GARandomBit())
    pop->individual( i ).copy(*mom);
      else
    pop->individual( i ).copy(*dad);
    }
    stats.nummut += (mut = pop->individual( i ).mutate(pMutation()));
    if(mut > 0) c1 = 1;

    stats.numeval += c1;
  }

  stats.numrep += pop->size();
  pop->evaluate(gaTrue);    // get info about current pop for next time

// If we are supposed to be elitist, carry the best individual from the old
// population into the current population.  Be sure to check whether we are
// supposed to minimize or maximize.

  if(minimaxi() == GAGeneticAlgorithm::MAXIMIZE) {
    if(el && oldPop->best().score() > pop->best().score())
      oldPop->replace(pop->replace(&(oldPop->best()), GAPopulation::WORST),
              GAPopulation::BEST);
  }
  else {
    if(el && oldPop->best().score() < pop->best().score())
```

```
      oldPop->replace(pop->replace(&(oldPop->best()), GAPopulation::WORST),
              GAPopulation::BEST);
  }
  stats.update(*pop);       // update the statistics by one generation
}
```

I.3 Building the GVR C++ source code

Because the function prototypes and bodies are all contained in the “.h” files rather than being broken up (lots of library writers are moving to this approach in the C++ community as of this writing), no full fledged Makefile is needed. Instead, the following Bash script will suffice just fine. It depends on the environment variables that point to the GALib static library and its include header files. BOOST is a C++ library available at `http://www.boost.org` that is only used to do some typecasting (from string to numeric values).

```
#!/bin/bash


g++ test_harness.cpp -I$GALIB_INC -I$BOOST32_INCLUDES \
 -L$GALIB_LIB -L$BOOST32_LIBS -lm -lga -o gvr
```

Bibliography

1. "Bin Packing and Machine Scheduling". URL `http://www.ams.org/new-in-math/cover/packings1.html`.

2. Aarts, Emile and Jan Karel Lenstra. *Local Search in Combinatorial Optimization.* Princeton University Press, 2003.

3. Abu-Ghazaleh, ANael, Richard Linderman, Robert Hillman, and James Hanna. "Exploiting HHPC for Parallel Discrete Event Simulation". *DoD High Performance Computing Modernization Program: Proceedings of the 2004 User's Group Conference*, 250–253. TO ADD IN, 2004.

4. Badeau, P., M. Gendreau, F. Guertin, J.-Y. Potvin, and É. D. Taillard. "A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows". *Transportation Research-C 5*, 109–122, 1997.

5. Bäck, T, D B Fogel, and T Michalewicz. *Evolutionary Computation 1.* Institute of Physics Publishing, 2000.

6. Bäck, T, D B Fogel, and Z Michalewicz. *Evolutionary Computation 2.* Institute of Physics Publishing, 2000.

7. Bonabeau, Eric, Marco Dorigo, and Guy Theraulaz. "Inspiration for Optimization from Social Insect Behaviour". *Nature*, 406:39–42, 2000.

8. Branch and Cut. "Vehicle Routing Data Sets". URL `http://branchandcut.org/VRP/data/`.

9. Braysy, Olli. "Genetic Algorithms for the Vehicle Routing Problem with Time Windows". *Arpakannus*, 33–38, 2001.

10. Bruegge, Bernd and Allen H. Dutoit. *Object-Oriented Software Engineering.* Prentice Hall, 2000.

11. Bullnheimer, Bernd, Gabriele Kotsis, and Christine Strauss. "Parallelization Strategies for the Ant System". Kluwer Series of Applied Optimization, 1997.

12. di Caro, Gianni and Marco Dorigo. "AntNet Distributed Stigmergetic Control for Communications Networks". *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

13. Casti, John L. *Reality Rules I: Picturing the World in Mathematics.* John Wiley and Sons, 1997.

14. Clarke, G. and J.W. Wright. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points". *Operations Research*, 12:568–581, 1964.

15. Cook, Stephen A. *The P Versus NP Problem.* Technical report, Clay Mathematics Institute, 2000.

16. Corner, Joshua. *Swarming Reconnaissance Using Unmanned Aerial Vehicles In A Parallel Discrete Event Simulation.* Master's thesis, Air Force Institute of Technology, 2004.

17. Corner, Joshua J. and Gary B. Lamont. "Parallel Simulation of UAV Swarm Scenarios". *Proceedings of the 2004 Winter Simulation Conference.* 2004.

18. Costa, D. and A. Hertz. "Ants Can Colour Graphs". *Journal of the Operational Research Society*, 48:295–305, 1997.

19. Crichton, Michael. *Prey.* Avon, 2003.

20. Czech, Zbigniew J. and Piotr Czarnas. “Parallel Simulated Annealing for the Vehicle Routing Problem with Time Windows”. *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, Canary Islands - Spain, (January 9-11, 2002):376–383, 2002.

21. Dasgupta, D., S. Yu, and N.S. Majumdar. “MILA - Multilevel Immune Learning Algorithm”. *Genetic and Evolutionary Computation Conference (GECCO)*. 2003.

22. Dasgupta, Dipankar. “Artificial Neural Networks and Artificial Immune Systems: Similarities and Differences”. *Sixth International Conference on Systems, Man, and Cybernetics*, 363–374. 1997.

23. Davis, L. “Applying Adaptive Algorithms to Epistatic Domains”. *Proceedings of the International Joint Conference on Artificial Intelligence*, 162–164. 1995.

24. Dorigo, Marco. *Ottimizzazione, apprendimento automatico, ed algoritmi basati su metafora naturale (Optimization, Learning and Natural Algorithms)*. Ph.D. thesis, Politecnico di Milano, Italy, 1992.

25. Eric Bonabeau and Marco Dorigo and Guy Theraulaz. *Swarm Intelligence*. Oxford University Press, 1999.

26. Fogel, D.B. “An Evolutionary Approach to the Traveling Salesman Problem”. *Biological Cybernetics*, 60:139–144, 1988.

27. Fujimoto, Richard M. *Parallel and Distributed Simulation Systems*. John Wiley and Sons, 2000.

28. Gambardella, L. M. and M. Dorigo. “Ant-Q A Reinforcement Learning Approach to the Travelling Salesman Problem”. *Twelfth International Conference on Machine Learning, ML-95*, 1995.

29. Gambardella, L.M., A. E. Rizzoli, F. Oliverio, N. Casagrande, A. V. Donati, R. Montemanni, and E. Lucibello. “Ant Colony Optimization for Vehicle Routing in Advanced Logistics Systems”. *Proceedings of the International Workshop on Modelling and Applied Simulation (MAS 2003)*, 3–9. 2–3.

30. Garey, M. R. and D.S. Johnson. *Computers and Intractibility*. W. H. Freeman, 1979.

31. Goldberg, D.E. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, 1989.

32. Goldberg, D.E. and R. Lingle. “Proceedings of the First International Conference on Genetic Algorithms”. J. J. Grefenstette (editor), *Alleles, Loci and the TSP*, 154–159. Lawrence Erlbaum Associates, 1985.

33. Grama, Anath, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to Parallel Computing*. Addison Wesley, second edition edition, 2003.

34. H.Cormen, Thomas. *Introduction to Algorithms*. The MIT Press, 2002.

35. Hofstadter, Douglas R. *Godel, Escher, Bach: An Eternal Golden Braid*. Basic Book, 1999.

36. Holland, John H. *Adaptationin Natural and Artificial Systems*. Bradford Books, 1975.

37. Homberger, Jörg and Hermann Gehring. “Two Evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows”. *INFOR*, 37(3):297 – 318, August 1999.

38. IEEE. *Proposed IEEE Standard Draft for Information Technology- Protocols for Distributed Interactive Simulation Applications*. Technical report, IEEE, 1993.

39. Kadrovach, Tony. *Communications Modeling System For Swarm-Based Sensors*. Ph.D. thesis, Air Force Institute of Technology, 2003.

40. Kleeman, Mark. *Evaluation and Optimization of UAV Swarm Multi-Objectives*. Master's thesis, Air Force Institute of Technology, 2004.

41. van Leeuwen, J. "Algorithmic Modeling and Complexity (Lecture Notes)", Fall 2003. URL `http://www.cs.uu.nl/docs/vakken/amc/lecture03-2.pdf`.

42. Levine, John and Frederick Ducatelle. "Ant Colony Optimisation and Local Search for Bin Packing and Cutting Stock Problems". 2002.

43. Machado, Penousal, Jorge Tavares, Francisco B. Pereira, and Ernesto Costa. *Vehicle Routing Problem doing it the Evolutionary Way*. Technical report, Instituto Superior de Engenharia De Coimbra, 2002.

44. Metron, INC. *SPEEDES's Users Guide*. Http//www.speedes.com.

45. Michalewicz, Zbigniew and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2002.

46. Mitchell, Tom. *Machine Learning*. McGraw Hill, 1997.

47. Nurcahyo, Gunadi W., Rose Alinda Alias, Siti Maryam Shamsuddin, and Mohd. Noor Md. Sap. "Vehicle Routing Problem For Public Transport A Case Study". 2000.

48. Ochi, Luiz S., Dalessandro S. Vianna, Lucia M. A. Drummond, and Andre O. Victor. "A Parallel Evolutionary Algorithm for the Vehicle Routing Problem with Heterogeneous Fleet". 1998.

49. Oliver, I.M., D.J. Smith, and J.R.C. Holland. "A Study of Permutation Crossover Operators on the Traveling Salesman Problem". J. J. Grefenstette (editor), *Proceedings of the 2nd International Converence on Genetic Algorithms*, 224–230. 1987.

50. Pae, Peter. "Unmanned Aircraft Gaining The Pentagon's Confidence". URL `http//ebird/afis.osd.mil/ebfiles/e20040809310502.html`.

51. Pereira, Francisco B., Jorge Tavares, Penousal Machado, and Ernesto Costa. *GVR a New Genetic Representation for the Vehicle Routing Problem*. Technical report, Instituto Superior de Engenharia De Coimbra, 2002.

52. Ralphs, T.K., L. Kopman, W.R. Pulleyblank, and L.E. Trotter Jr. "On the Capacitated Vehicle Routing Problem". *Mathematical Programming*, Series B, 2003.

53. Reynolds, Craig W. "Flocks, Herds, and Schools: A Distributed Behavioral Model". *Computer Graphics*, 21(4):25–34, 1987.

54. Richardson, JT, MR Palmer, G Liepins, and M Hilliard. "Some Guidelines for Genetic Algorithms with Penalty Functions". JD Schaffer (editor), *Proc. 3rd Int. Conf. on Genetic Algorithms*, 191–197. Morgan Kaufmann, 1989.

55. Russell, Stuart and Peter Norvig. *Artificial Intelligence A Modern Approach*. Prentice Hall, 2003.

56. Schmidt, Douglas C. "Overview of ACE". Http//www.cs.wustl.edu/˜ schmidt/ACE-overview.html.

57. Schoonderwoerd, Ruud, Owen Holland, Janet Bruten, and Leon Rothkrantz. "Ant-based Load Balancing in Telecommunications Networks". HP Labs Technical Report (HPL-96-76) 1996.

58. Sim, Kwang Mong and Weng Hong Sun. "Ant Colony Optimization for Routing and Load-Balancing Survey and New Directions". *IEEE Transactions on Systems, Man and Cybernetics-Part A Systems and Humans*, 33, 2003.

59. Society, IEEE Computer. "Dialog Modeling and Semsor Networks", 2004.

60. Sun, Xian-He H. and Lionel M. Ni. "Scalable Problems and Memory-Bounded Speedup". *Journal of Parallel and Distributed Computing*, 19(1):27–37, 1993.

61. Taillard, Éric, Phillipe Badeau, Michel Gendreau, Francois Guertin, and Jean-Yves Potvin. *A Tabu Search Heristic for the Vehicle Routing Problem with Soft Time Windows*. Technical Report CRT-95-66, CRT, 1995.

62. Tavares, Jorge, Penousal Machado, Francisco B. Pereira, and Ernesto Costa. *Crossover and Diversity A Study about GVR*. Technical report, Instituto Superior de Engenharia De Coimbra, 2003.

63. Tavares, Jorge, Penousal machado, Francisco B. Pereira, and Ernesto Costa. "On the Influence of GVR in Vehicle Routing". *Symposium on Applied Computing*. 2003.

64. Tavares, Jorge, Francisco B. Pereira, Penousal Machado, and Ernesto Costa. "GVR Delivers It On Time". *4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL'02)*, 745–749. 2002.

65. Thangiah, Sam R. "An Adaptive Clustering Method using a Geometric Shape for Vehicle Routing Problems with Time Windows". *Proceedings of the 6th International Conference on Genetic Algorithms*, 536–543. Morgan Kaufmann, 1995.

66. Thiébaut, D. "Parallel Programming in C for the Transputer". URL `http://cs.smith.edu/{\~~}thiebaut/transputer/chapter8/chap8-2%.html`.

67. Thornton, Chris. "The Building Block Fallacy". *Complexity International*, 1997.

68. Toth, Paolo and Daniele Vigo (editors). *The Vehicle Routing Problem*. SIAM, 2002.

69. of Transportation Technology, Department and National Chiao Tung University Management. "Transportation Network Laboratory". URL `http://www.tem.nctu.edu.tw/network/`.

70. University, Carnegie Mellon. "A Low Cost Unmanned Airr Vehicle". URL `http://gs3636.sp.cs.cmu.edu/uav/index.html`.

71. Walls, Matthew. "GAlib: A C++ Library of Genetic Algorithm Components". URL `http://lancet.mit.edu/ga/`.

72. Whitley, Darrell. "A Genetic Algorithm Tutorial". 1994.

73. Wolpert, David H. and William G. Macready. "No Free Lunch Theorems for Optimization". *IEEE Transactions on Evolutionary Computation*, 19(1):67–82, April 1997.

74. Zecchin, Aaron, Holger Maier, Angus Simpson, Andrew Roberts, Matthew Berrisford, and Michael Leonard. "Max-Min Ant System Applied to Water Distribution System Optimisation". *MODSIM 2003 International Congress on Modelling and Simulation in Townsville, Queensland, Australia*, 795–800. 2003.

75. Zhu, Kenny Qili. "A New Genetic Algorithm For VRPTW". *International Conference on Artificial Intelligence, Las Vegas, USA*. 2000.

REPORT DOCUMENTATION PAGE		*Form Approved* *OMB No. 074-0188*
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**		
1. REPORT DATE *(DD-MM-YYYY)* 03-21-2005	**2. REPORT TYPE** **Master's Thesis**	**3. DATES COVERED** *(From – To)* Aug 2003 – Mar 2005

4. TITLE AND SUBTITLE A Genetic Algorithm for UAV Routing Integrated with a Parallel Swarm Simulation	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Russell, Matthew A., 2Lt, USAF	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/ENG) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCS/ENG/05-16
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/SNZW Attn: Mr. Mike Foster 2241 Avionics Circle WPAFB OH 45433-7303 DSN: 986-4899 x3030	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This research investigation addresses the problem of routing and simulating swarms of UAVs. Sorties are modeled as instantiations of the NP-Complete Vehicle Routing Problem, and this work uses genetic algorithms (GAs) to provide a fast and robust algorithm for a priori and dynamic routing applications. Swarms of UAVs are modeled based on extensions of Reynolds' swarm research and are simulated on a Beowulf cluster as a parallel computing application using the Synchronous Environment for Emulation and Discrete Event Simulation (SPEEDES). In a test suite, standard measures such as benchmark problems, best published results, and parallel metrics are used as performance measures.

The GA consistently provides efficient and effective results for a variety of VRP benchmarks. Analysis of the solution quality over time verifies that the GA exponentially improves solution quality and is robust to changing search landscapes–making it an ideal tool for employment in UAV routing applications. Parallel computing metrics calculated from the results of a PDES show that consistent speedup (almost linear in many cases) can be obtained using SPEEDES as the communication library for this UAV routing application. Results from the routing application and parallel simulation are synthesized to produce a more advanced model for routing UAVs.

15. SUBJECT TERMS
Unmanned Aerial Vehicle, Vehicle Routing, Genetic Algorithm, Evolutionary Computation, High Performance Computing, Parallel Discrete Event Simulation, Swarm Intelligence

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dr. Gary B. Lamont (AFIT/ENG)
REPORT U	**ABSTRACT** U	**c. THIS PAGE** U	UU	240	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636, ext 4718; e-mail: gary.lamont@afit.edu